Principia Softwarica: The Web Browser **`mmm`**

version 0.2

Yoann Padioleau
`yoann.padioleau@gmail.com`

with code from
Francois Rouaix

February 25, 2026

Contents

1 Introduction **13**
- 1.1 Motivations . . . 13
- 1.2 `mmm` . . . 13
- 1.3 Other Web browsers . . . 13
- 1.4 Getting started . . . 13
- 1.5 Requirements . . . 13
- 1.6 About this document . . . 13
- 1.7 Copyright . . . 14
- 1.8 Acknowledgments . . . 14

2 Overview **15**
- 2.1 Web browser principles . . . 15
- 2.2 `mmm` services . . . 15
- 2.3 HTML document language . . . 16
- 2.4 `hello.html` . . . 16
- 2.5 CSS style language . . . 16
- 2.6 `hello.css` . . . 16
- 2.7 Javascript scripting language . . . 16
- 2.8 `hello.js` . . . 16
- 2.9 Code organization . . . 16
- 2.10 Software architecture . . . 16
  - 2.10.1 Trace of a web request . . . 16
  - 2.10.2 Trace of a mouse click . . . 16
- 2.11 Book structure . . . 16

3 Core Data Structures **17**
- 3.1 URLs and co . . . 17
  - 3.1.1 URLs and protocols . . . 17
  - 3.1.2 URIs and fragments . . . 17
  - 3.1.3 Hypertext Links . . . 18
  - 3.1.4 Web requests . . . 18
- 3.2 Documents . . . 19
  - 3.2.1 Document ID . . . 20
  - 3.2.2 Document cache . . . 20
- 3.3 Protocols and document flow . . . 21
  - 3.3.1 `Protos.protos` . . . 21
  - 3.3.2 Document continuation . . . 22
  - 3.3.3 Document handle . . . 22
- 3.4 HTTP . . . 22
  - 3.4.1 Requests . . . 22

3.4.2 Responses . . . 23
3.4.3 Headers and content types . . . 23
3.5 Viewers . . . 23
3.5.1 `Viewers.viewers` . . . 23
3.5.2 `Viewers.context` . . . 24
3.5.3 `Viewers.display_info` . . . 25
3.6 Abstract syntax trees . . . 25
3.6.1 HTML . . . 25
3.6.2 CSS . . . 26
3.6.3 Javascript . . . 26
3.7 DOM . . . 26
3.8 HTML display . . . 26
3.8.1 Formatter . . . 26
3.8.2 Interpreter . . . 27
3.9 Summary . . . 28

4 `main()` 29
4.1 Initialisations . . . 30
4.1.1 Graphics initialisation . . . 30
4.1.2 Resources initialisation . . . 31
4.1.3 Local initialisation . . . 31
4.1.4 Initial fake URL and HTML document . . . 31
4.2 `Mmm.initial_navigator()` . . . 33
4.2.1 Initial URL . . . 33
4.2.2 `Mmm.navigator()` and `Nav.t` . . . 34
4.2.3 `Nav.absolutegoto()` . . . 35
4.2.4 `Nav.request()` . . . 36
4.2.5 `Nav.process_viewer()` . . . 37
4.2.6 `Mmm.show_current()` . . . 37
4.2.7 `make_ctx()` . . . 38
4.2.8 XXX . . . 40
4.3 The event loop . . . 40

5 Navigator Interface 41
5.1 Layout . . . 41
5.2 Address bar . . . 42
5.3 Canvas . . . 42
5.4 History . . . 42
5.4.1 Back button . . . 43
5.4.2 Forward button . . . 43
5.5 Home . . . 43
5.6 Menus . . . 44
5.6.1 MMM menu . . . 44
5.6.2 Navigation menu . . . 46
5.6.3 Document menu . . . 46
5.6.4 Other menu . . . 47
5.6.5 Help menu . . . 47
5.7 Status log . . . 47
5.8 Keyboard shortcuts . . . 48
5.9 Events . . . 49

6 Parsing **51**
6.1 URLs . . . 51
6.1.1 Protocol . . . 51
6.1.2 Host, port . . . 52
6.1.3 Path, search . . . 53
6.1.4 Normalization . . . 53
6.1.5 Encoding . . . 55
6.2 Links . . . 57
6.3 HTML . . . 59
6.3.1 Lexing . . . 59
6.3.2 Parsing part 1, the DTD . . . 68
6.3.3 Parsing part 2, the SGML evaluator . . . 75
6.3.4 `htparse` . . . 81
6.4 HTTP . . . 84
6.4.1 Request . . . 84
6.4.2 Status . . . 84
6.4.3 Headers . . . 86

7 Retrieving **89**
7.1 `Retrieve.f()` . . . 89
7.2 Behaviour . . . 90
7.2.1 `Retrieve.http_check()` . . . 90
7.2.2 HTTP status codes . . . 91
7.3 Active connections . . . 92
7.4 Connections, `Feed.t` . . . 93
7.5 Logging . . . 95
7.6 Images . . . 96
7.7 Scheduling . . . 97
7.8 Progressing . . . 97

8 HTTP **98**
8.1 The request . . . 98
8.1.1 `Http.req()` . . . 98
8.1.2 `Http.tcp_connect()` . . . 99
8.1.3 `Http.start_request()` . . . 102
8.1.4 `Http.full_request()` . . . 102
8.1.5 `Http.failed_request()` . . . 104
8.2 The response . . . 104
8.2.1 `Http.process_response()` . . . 104
8.2.2 Reading headers . . . 105
8.2.3 Error management . . . 106
8.2.4 Timeout management . . . 107
8.3 Headers merging . . . 107
8.4 Time . . . 108
8.5 HTTPS and TLS and SSL . . . 110

9 Viewers **111**
9.1 `Viewers.f()` . . . 111
9.1.1 Dispatching . . . 112
9.1.2 Error management . . . 112

9.2 Content types . . . 113
9.2.1 text/plain, Plain.viewer() . . . 113
9.2.2 text/html, Htmlw.viewer() . . . 115
9.3 MIME types . . . 116
9.3.1 Http_headers.suffixes . . . 116
9.3.2 mmm -suffixes . . . 117
9.3.3 Filename suffix to content type . . . 118
9.3.4 Retype.f() . . . 118
9.4 External viewer . . . 118
9.5 Interactive viewer . . . 120
9.6 Save to file viewer . . . 121

10 HTML Display 122
10.1 The display info, Htmlw.display_html . . . 122
10.2 The display machine, Html_disp.machine . . . 125
10.2.1 class Html_disp.display_machine . . . 125
10.2.2 Html_disp.Make.init() . . . 129
10.3 The formatter . . . 129
10.4 HTML attributes . . . 133
10.5 HTML tags . . . 134
10.5.1 Header . . . 134
10.5.2 Content, <body> . . . 137
10.5.3 Headings, <h1>, <h2>, etc . . . 137
10.5.4 Paragraphs, <p> . . . 138
10.5.5 Text styles part 1 . . . 139
10.5.6 Text styles part 2 . . . 141
10.5.7 Text styles part 3 . . . 142
10.5.8 Text styles part 4 . . . 143
10.5.9 Lists . . . 144
10.5.10 Anchors, <a> . . . 148
10.5.11 Breaks . . . 150
10.5.12 Tables . . . 152
10.5.13 Maps, <map> . . . 152
10.5.14 Fonts . . . 155
10.5.15 <style> . . . 157
10.5.16 <script> . . . 158
10.5.17 Frames . . . 158
10.6 XXX . . . 159
10.6.1 Annotations . . . 159
10.6.2 Extra headers . . . 159
10.6.3 Error management . . . 160

11 Embedded Elements 161
11.1 Overview . . . 161
11.2 XXX . . . 161
11.3 Embedded viewers . . . 164

12 Images 169
12.1 The image loader . . . 171
12.2 convert fallback . . . 172

13 Forms **173**

14 CSS **174**
14.1 Lexing . . . 174
14.2 Parsing . . . 174
14.3 Evaluating . . . 174

15 Javascript **175**
15.1 Lexing . . . 175
15.2 Parsing . . . 175
15.3 Evaluating . . . 175

16 Applets **176**
16.1 `hello_applet.cmo` . . . 176
16.2 Data structures . . . 176
16.3 Initialisation . . . 176
16.4 Viewers . . . 178
16.4.1 Regular viewer . . . 178
16.4.2 Embed viewer . . . 179
16.5 Loading OCaml bytecode: `Dynlink` . . . 180
16.5.1 `mod_cache` . . . 180
16.5.2 `Dload.load()` . . . 182
16.5.3 Applet kind . . . 182
16.5.4 Low-level unsafe load . . . 183
16.5.5 Registered functions . . . 185
16.5.6 Pending loads . . . 185
16.6 Calling OCaml bytecode . . . 186
16.7 Sandboxing . . . 186
16.8 Capabilities . . . 187
16.8.1 Paranoid setting . . . 187
16.9 Advanced features . . . 187
16.9.1 Applet preferences . . . 187
16.9.2 Applet user menu . . . 187
16.9.3 Machine hooks . . . 188
16.9.4 External windows . . . 188
16.9.5 Initial modules and `mmm -nomodule` . . . 188
16.9.6 Loading OCaml Source . . . 189
16.9.7 Loading signed bytecode . . . 190

17 Web-developer tools **192**
17.1 View source . . . 192

18 Concurrency **196**
18.1 Tasks . . . 196
18.2 Tachymeter . . . 196
18.2.1 backend . . . 196
18.2.2 frontend . . . 197

19 Extra features **199**
19.1 Fragment navigation . . . 199
19.2 History navigation . . . 200
19.3 Lifecycle control . . . 202
19.3.1 Abort . . . 202
19.3.2 Refresh . . . 203
19.3.3 Redisplay . . . 204
19.4 Progress feedback . . . 205
19.5 Bookmarks . . . 205
19.6 Hotlist . . . 205
19.7 Multiple windows . . . 206
19.8 User preferences . . . 207
19.8.1 Preference file, `mmm -prefs` . . . 207
19.8.2 Initial geometry, `mmm -geometry` . . . 208
19.9 TK customization . . . 208
19.9.1 Color palette, `mmm -palette` . . . 208
19.9.2 Click to follow, `mmm -clicktofocus` . . . 209
19.10 Help URL, `mmm -helpurl` . . . 209
19.11 Extra Protocols . . . 209
19.11.1 `file://` . . . 209
19.11.2 `mailto://` . . . 213
19.11.3 Proxied protocols . . . 215
19.11.4 `ftp://` . . . 215
19.11.5 `telnet://` . . . 216
19.11.6 `nntp://` . . . 216
19.11.7 Old protocols . . . 216
19.12 Gzip decoders . . . 217
19.13 Animaged GIFs . . . 218
19.14 Audio . . . 219
19.15 video . . . 219
19.16 `mmm_remote` . . . 219
19.16.1 Server . . . 219
19.16.2 Client . . . 219

20 Advanced Topics **221**
20.1 Proxy . . . 221
20.2 Forwarding . . . 222
20.3 Encodings . . . 222
20.3.1 Base 64 . . . 222
20.4 i18n . . . 224
20.4.1 Special characters . . . 227
20.5 Referer . . . 228
20.6 Security . . . 229
20.6.1 Parsing . . . 230
20.6.2 Security challenge . . . 230
20.6.3 HTTP status . . . 234
20.7 Optimisations . . . 235
20.7.1 Document cache, `Cache` . . . 235
20.7.2 Graphic cache, `Gcache` . . . 237
20.8 Frames . . . 238

20.9 Fake CGI . . . 239
20.10Signals . . . 240

21 Conclusion **242**

A Debugging **243**
A.1 Logging . . . 243
A.2 Postmortem . . . 243
A.3 Subsystems . . . 244
A.3.1 Requests . . . 244
A.3.2 Documents . . . 244
A.3.3 HTML . . . 245
A.3.4 HTTP . . . 245
A.3.5 Caches . . . 246
A.3.6 Viewer . . . 246
A.3.7 Scheduler . . . 246
A.3.8 HTML display . . . 246
A.3.9 GUI . . . 246
A.4 Dumpers . . . 247
A.4.1 URLs . . . 247
A.4.2 Links . . . 249
A.4.3 DTDs . . . 249
A.4.4 HTML . . . 249

B Profiling **250**

C Error Management **251**
C.1 `Error.t` . . . 251
C.2 Subsystems . . . 252
C.2.1 Requests . . . 252
C.3 Exceptions . . . 252
C.3.1 URL . . . 252
C.3.2 Links . . . 252
C.3.3 Web requests . . . 252
C.3.4 HTML . . . 252
C.3.5 HTTP . . . 252

D A Preferences Library **253**

E Standard Library **254**
E.1 Lists . . . 254
E.2 Strings . . . 255
E.3 Extensible buffers . . . 258
E.4 Files . . . 259
E.5 Dates . . . 261

F Extra Code **263**
F.1 `core/` . . . 263
F.1.1 `core/condition.mli` . . . 263
F.1.2 `core/condition.ml` . . . 264
F.1.3 `misc/date.mli` . . . 264

F.1.4 misc/date.ml . . . 265
F.1.5 misc/ebuffer.mli . . . 265
F.1.6 misc/ebuffer.ml . . . 265
F.1.7 misc/error.mli . . . 266
F.1.8 misc/error.ml . . . 266
F.1.9 misc/i18nprintf.mli . . . 266
F.1.10 misc/i18nprintf.ml . . . 268
F.1.11 misc/ibtree.mli . . . 271
F.1.12 misc/ibtree.ml . . . 271
F.1.13 i18n/lang.mli . . . 274
F.1.14 i18n/lang.ml . . . 274
F.1.15 misc/log.mli . . . 274
F.1.16 misc/log.ml . . . 275
F.1.17 core/low.mli . . . 275
F.1.18 core/low.ml . . . 276
F.1.19 misc/mlist.mli . . . 279
F.1.20 misc/mlist.ml . . . 279
F.1.21 misc/mstring.mli . . . 279
F.1.22 misc/mstring.ml . . . 280
F.1.23 misc/msys.mli . . . 280
F.1.24 misc/msys.ml . . . 281
F.1.25 misc/i18n.mli . . . 281
F.1.26 misc/i18n.ml . . . 281
F.1.27 misc/munix.ml . . . 282
F.1.28 misc/feed.mli . . . 284
F.1.29 misc/feed.ml . . . 285
F.1.30 tk/glevents.mli . . . 285
F.1.31 tk/glevents.ml . . . 285
F.1.32 misc/hotlist.ml . . . 285
F.1.33 globals/version.mli . . . 285
F.1.34 globals/version.ml . . . 286
F.2 www/ . . . 286
F.2.1 www/uri.mli . . . 286
F.2.2 www/uri.ml . . . 286
F.2.3 www/url.mli . . . 286
F.2.4 www/url.ml . . . 287
F.2.5 www/urlenc.mli . . . 287
F.2.6 www/urlenc.ml . . . 287
F.2.7 www/lexurl.mli . . . 288
F.2.8 www/lexurl.mll . . . 288
F.2.9 www/hyper.mli . . . 288
F.2.10 www/hyper.ml . . . 289
F.2.11 www/maps.mli . . . 289
F.2.12 www/maps.ml . . . 290
F.2.13 www/www.mli . . . 290
F.2.14 www/www.ml . . . 291
F.2.15 www/document.mli . . . 291
F.2.16 www/document.ml . . . 291
F.3 html/ . . . 292

F.3.1 html/dtd.mli . . . 292
F.3.2 html/dtd.ml . . . 293
F.3.3 html/html.mli . . . 293
F.3.4 html/html.ml . . . 294
F.3.5 html/lexhtml.mli . . . 296
F.3.6 html/lexhtml.mll . . . 296
F.3.7 html/html_eval.mli . . . 297
F.3.8 html/html_eval.ml . . . 297
F.3.9 html/htparse.ml . . . 298
F.4 http/ . . . 298
F.4.1 http/base64.mli . . . 298
F.4.2 http/base64.ml . . . 298
F.4.3 http/http_date.mli . . . 299
F.4.4 http/http_date.ml . . . 299
F.4.5 http/messages.mli . . . 300
F.4.6 http/http_headers.mli . . . 300
F.4.7 http/http_headers.ml . . . 301
F.4.8 http/auth.mli . . . 303
F.4.9 http/auth.ml . . . 303
F.4.10 http/lexheaders.mli . . . 304
F.4.11 http/lexheaders.mll . . . 304
F.4.12 http/retype.mli . . . 306
F.4.13 http/retype.ml . . . 306
F.4.14 http/http.mli . . . 306
F.4.15 http/http.ml . . . 306
F.5 protocols/ . . . 308
F.5.1 protocols/cache.mli . . . 308
F.5.2 protocols/cache.ml . . . 310
F.5.3 protocols/file.mli . . . 316
F.5.4 protocols/file.ml . . . 316
F.5.5 protocols/mailto.ml . . . 317
F.5.6 protocols/protos.mli . . . 317
F.5.7 protocols/protos.ml . . . 318
F.6 retrieve/ . . . 318
F.6.1 retrieve/progress.mli . . . 318
F.6.2 retrieve/progress.ml . . . 318
F.6.3 retrieve/retrieve.mli . . . 318
F.6.4 retrieve/retrieve.ml . . . 318
F.6.5 retrieve/img.mli . . . 320
F.6.6 retrieve/img.ml . . . 320
F.6.7 retrieve/scheduler.mli . . . 323
F.6.8 retrieve/scheduler.ml . . . 324
F.7 viewers/ . . . 332
F.7.1 viewers/decoders.mli . . . 332
F.7.2 viewers/decoders.ml . . . 333
F.7.3 viewers/embed.mli . . . 333
F.7.4 viewers/embed.ml . . . 333
F.7.5 viewers/save.mli . . . 335
F.7.6 viewers/save.ml . . . 335

F.7.7 viewers/viewers.mli . . . 339
F.7.8 viewers/viewers.ml . . . 340
F.7.9 viewers/plain.ml . . . 342
F.8 display/ . . . 342
F.8.1 display/attrs.mli . . . 342
F.8.2 display/attrs.ml . . . 345
F.8.3 display/cmap.ml . . . 352
F.8.4 display/fit.mli . . . 354
F.8.5 display/fit.ml . . . 355
F.8.6 display/fonts.mli . . . 360
F.8.7 display/fonts.ml . . . 361
F.8.8 display/hr.mli . . . 365
F.8.9 display/hr.ml . . . 365
F.8.10 display/htbind.mli . . . 365
F.8.11 display/htbind.ml . . . 367
F.8.12 display/source.mli . . . 370
F.8.13 display/source.ml . . . 370
F.8.14 display/ctext.mli . . . 370
F.8.15 display/ctext.ml . . . 371
F.8.16 display/htmlfmt.ml . . . 374
F.8.17 display/imgload.mli . . . 375
F.8.18 display/imgload.ml . . . 376
F.8.19 display/html_form.mli . . . 382
F.8.20 display/html_form.ml . . . 383
F.8.21 display/htmlw.mli . . . 387
F.8.22 display/htmlw.ml . . . 387
F.8.23 display/form.mli . . . 394
F.8.24 display/form.ml . . . 395
F.8.25 display/html_table.mli . . . 402
F.8.26 display/html_table.ml . . . 402
F.8.27 display/styles.mli . . . 405
F.8.28 display/styles.ml . . . 405
F.8.29 display/table.mli . . . 407
F.8.30 display/table.ml . . . 408
F.8.31 display/htframe.ml . . . 416
F.8.32 display/textw_fo.mli . . . 421
F.8.33 display/textw_fo.ml . . . 422
F.8.34 display/html_disp.mli . . . 423
F.8.35 display/html_disp.ml . . . 423
F.9 gui/ . . . 423
F.9.1 gui/about.mli . . . 423
F.9.2 gui/about.ml . . . 424
F.9.3 gui/fontprefs.ml . . . 427
F.9.4 gui/gcache.mli . . . 431
F.9.5 gui/gcache.ml . . . 432
F.9.6 gui/plink.mli . . . 434
F.9.7 gui/plink.ml . . . 434
F.9.8 gui/prefs.mli . . . 435
F.9.9 gui/prefs.ml . . . 438

F.9.10 gui/debug.ml . . . 448
F.9.11 gui/history.mli . . . 448
F.9.12 gui/history.ml . . . 448
F.9.13 gui/nav.mli . . . 450
F.9.14 gui/nav.ml . . . 450
F.9.15 gui/mmmprefs.mli . . . 454
F.9.16 gui/mmmprefs.ml . . . 455
F.9.17 gui/mmm.mli . . . 459
F.9.18 gui/mmm.ml . . . 460
F.9.19 gui/cci.ml . . . 461
F.10 main/ . . . 463
F.10.1 main.ml . . . 463
F.10.2 main_remote.ml . . . 464
F.11 extensions/ . . . 464
F.11.1 extensions/audio.ml . . . 464
F.11.2 extensions/images.ml . . . 464
F.11.3 extensions/remove_simple_table.ml . . . 466
F.11.4 extensions/tachy_aftermmm.ml . . . 468
F.11.5 extensions/tachy_space.ml . . . 473
F.11.6 extensions/tachy_test.ml . . . 478
F.12 applets/ . . . 482

Glossary **496**

Indexes **497**

References **497**

Chapter 1

Introduction

1.1 Motivations

The goal of this book is to present with full details the source code of a web browser. Why? Because I think you are a better programmer if you fully understand how things work under the hood.

1.2 `mmm`

1.3 Other Web browsers

Here are other candidates that were considered but ultimately discarded:

- nexus
- mosaic
- gecko (firefox)
- khtml/webkit/blink (kconqueror, safari, chrome)
- netsurf
- servo

1.4 Getting started

1.5 Requirements

1.6 About this document

This document is a *literate program* [Knu92]. It derives from a set of files processed by a tool, `syncweb` [Pad09], generating either this book or the actual source code of the program. The code and its documentation are thus strongly connected.

1.7 Copyright

Most of this document is actually source code from MMM, so those parts are copyright by INRIA.

⟨*copyright header v6* 14a⟩≡ (300a 299 298 287c 281a 265)

```
(***********************************************************************)
(*                                                                     *)
(*                           The V6 Engine                             *)
(*                                                                     *)
(*          Francois Rouaix, projet Cristal, INRIA Rocquencourt        *)
(*                                                                     *)
(*  Copyright 1996 Institut National de Recherche en Informatique et   *)
(*  Automatique.  Distributed only by permission.                      *)
(*                                                                     *)
(***********************************************************************)
```

⟨*copyright header calves* 14b⟩≡ (495 493 489c 483a)

```
(***********************************************************************)
(*                                                                     *)
(*                           Calves                                    *)
(*                                                                     *)
(*          Francois Rouaix, projet Cristal, INRIA Rocquencourt        *)
(*                                                                     *)
(*  Copyright 1996 Institut National de Recherche en Informatique et   *)
(*  Automatique.  Distributed only by permission.                      *)
(*                                                                     *)
(***********************************************************************)
```

The prose is mine and is licensed under the GNU Free Documentation License.

1.8 Acknowledgments

I would like to thank of course Francois Rouaix, the main author of MMM.

Chapter 2

Overview

2.1 Web browser principles

2.2 mmm services

⟨*constant* Main.usage_str 15a⟩≡ (463b)
```
  let usage_str =
    "Usage: meuh <opts> <initial url>"
```

⟨*signature* Version.http 15b⟩≡ (285e)
```
  val http : string              (* the version in User-Agent field *)
```

⟨*constant* Version.http 15c⟩≡ (286a)
```
  (* User-Agent field *)
  let http = "MMM/0." ^ string_of_int number
```

⟨*signature* Version.number 15d⟩≡ (285e)
```
  (* Version and other builtin strings *)
  val number : int
```

⟨*constant* Version.number 15e⟩≡ (286a)
```
  (* Version *)
  let number = 418
```

⟨*signature* Version.about 15f⟩≡ (285e)
```
  val about : string -> string   (* dialog *)
```

⟨*function* Version.about 15g⟩≡ (286a)
```
  (* dialog uses an gigantic font ! *)
  let about = function
    | "iso8859" ->
  "MMM Version 0." ^ version_number ^
  "\nWritten by Fran\231ois Rouaix
  Contributions by Jun P. Furuse and Jacques Garrigue
  Ported to O'Caml 3 by Jun P. Furuse and Pierre Weis
  \169 Copyright INRIA

  Projet Cristal
  INRIA Rocquencourt
  Domaine de Voluceau
  78153 Le Chesnay Cedex
  France

  Francois.Rouaix@inria.fr
  http://pauillac.inria.fr/~rouaix/
  "
    | s -> failwith (Printf.sprintf "language not supported: %s" s)
```

⟨*constant* `Version.version_number` 16⟩≡ (286a)

```
let version_number =
  string_of_int number
```

2.3 HTML document language

2.4 `hello.html`

2.5 CSS style language

2.6 `hello.css`

2.7 Javascript scripting language

2.8 `hello.js`

2.9 Code organization

2.10 Software architecture

2.10.1 Trace of a web request

2.10.2 Trace of a mouse click

2.11 Book structure

Chapter 3

Core Data Structures

3.1 URLs and co

3.1.1 URLs and protocols

⟨*type* Url.t 17a⟩≡ (287a 286d)
```
(* URLs as defined by RFC 1738. Not all components are used for all protocols.
 * The order of the fields below correspond to the actual order in the string:
 *   <protocol>://<user>:<password>@<host>:<port>/<path>?<search>
*)
type t =
  { mutable protocol : protocol;

    mutable user : string option;
    mutable password: string option;

    mutable host : string option;
    mutable port : int option;

    mutable path : string option;
    mutable search: string option
  }
```

⟨*type* Url.protocol 17b⟩≡ (287a 286d)
```
type protocol =
 | HTTP | HTTPS
 | FILE | FTP
 | MAILTO | NNTP
 | GOPHER | NEWS | WAIS | PROSPERO
 | TELNET
 | OtherProtocol of string
```

3.1.2 URIs and fragments

⟨*type* Uri.abs_uri 17c⟩≡ (286)
```
(* URI utilities. RFC 1630 *)
type abs_uri = {
   uri_url : string;
   uri_fragment : string option
 }
```

⟨*signature* Uri.is_absolute 17d⟩≡ (286b)
```
val is_absolute : string -> bool
   (* [is_absolute uri] determines if [uri] is absolute according to
      rules of RFC 1630 *)
```

⟨*function* Uri.is_absolute 18a⟩≡ (286c)
```
  (* RFC 1630, partial forms *)
  let is_absolute uri =
    try
      let colonpos = String.index uri ':' in
      try
        let slashpos = String.index uri '/' in
        colonpos < slashpos (* colon must occur before slash *)
      with
        Not_found -> true (* colon occurs before slash *)
    with
      Not_found -> false (* absolute must have a : *)
```

3.1.3 Hypertext Links

⟨*type* Hyper.link 18b⟩≡ (289a 288c)
```
  (* An hypertext(media) link on the Web *)
  type link = {
    h_uri : string;
    h_context: string option;

    h_method : link_method;  (* default is GET *)
    h_params : (string * string) list
  }
```

⟨*type* Hyper.link_method 18c⟩≡ (289a 288c)
```
  (* This is currently for HTTP and derived, but ... *)
  (* Contains only the one we support *)
  type link_method =
   | GET
   | POST of string
   ⟨Hyper.link_method other cases 104b⟩
```

⟨*signature* Hyper.default_link 18d⟩≡ (288c)
```
  val default_link: string -> link
```

⟨*function* Hyper.default_link 18e⟩≡ (289a)
```
  let default_link uri = {
   h_uri = uri;
   h_context = None;
   h_method = GET;
   h_params = [];
  }
```

3.1.4 Web requests

⟨*type* Www.request 18f⟩≡ (291a 290b)
```
  (*
   * Requests
   *)
  type request =  {
      www_link : Hyper.link;        (* the link that produced this request *)
      mutable www_headers : string list;    (* additional headers *)
      ⟨Www.request security field 229c⟩

      ⟨Www.request parsed link fields 19a⟩

      ⟨Www.request logging method 244d⟩
      ⟨Www.request error managment method 252a⟩
    }
```

⟨Www.request *parsed link fields* 19a⟩≡ (18f)
```
  www_url : Url.t;            (* parsed version *)
  www_fragment : string option; (* because viewer is passed down *)
```

⟨*signature* Www.make 19b⟩≡ (290b)
```
  val make : Hyper.link -> request
    (* raises: Url_Lexing | Invalid_link *)
```

⟨*function* Www.make 19c⟩≡ (291a)
```
  let make (hlink : Hyper.link) : request =
    let absuri = Hyper.resolve hlink in
    let url : Url.t = Lexurl.make absuri.uri_url in
    try (* search for space in network URI *)
      if List.mem url.protocol [FILE; MAILTO]
      then raise Not_found
      else
        let n =
          (* will raise Not_found if no space found *)
          Str.search_forward sp absuri.uri_url 0
        in
        raise (Hyper.Invalid_link (Hyper.UrlLexing ("suspicious white space", n)))
    with Not_found ->
      (* regular code path, everything is normal, we didn't find (bad) spaces *)
      { www_link = hlink;

        www_url = url; (* should not fail ? *)
        www_fragment = absuri.uri_fragment;

        www_auth = [];
        www_headers = [];

        www_logging = (fun _ -> ());
        www_error = !Error.default
      }
```

⟨*constant* Www.sp 19d⟩≡ (291a)
```
  let sp = Str.regexp "[ \t\n]"
```

3.2 Documents

⟨*type* Document.document 19e⟩≡ (291)
```
  type t = {
    document_headers : string list; (* e.g. Content-type: text/html *)
    mutable document_data : data;
    document_address : Url.t;
  }
```

⟨*type* Document.document_data 19f⟩≡ (291)
```
  (*
   * Information on a document, as could be requested by "other" clients,
   * that is clients not directly on the chain of processes dealing with
   * the handle
   *)
  type data =
   | MemoryData of Ebuffer.t
   | FileData of Fpath.t * bool (* flag is true if file is temporary *)
```

3.2.1 Document ID

⟨*type* Document.document_id 20a⟩≡ (291)
```
(* Document Id is a reference to a document in the browser.
   For some documents, e.g. results of POST queries, the URL is not a
   sufficient description. Stamp is 0 for unique documents.
*)
type id = {
  document_url : Url.t;
  document_stamp : int
}
```

⟨*signature* Document.no_stamp 20b⟩≡ (291b)
```
val no_stamp : int
```

⟨*constant* Document.no_stamp 20c⟩≡ (291c)
```
let no_stamp = 0
```

⟨*constant* Document.stamp_counter 20d⟩≡ (291c)
```
let stamp_counter = ref 0
```

⟨*signature* Document.new_stamp 20e⟩≡ (291b)
```
val new_stamp : unit -> int
```

⟨*function* Document.new_stamp 20f⟩≡ (291c)
```
let new_stamp () =
  incr stamp_counter; !stamp_counter
```

⟨*signature* Document.document_id 20g⟩≡ (291b)
```
val document_id : Www.request -> id
```

⟨*function* Document.document_id 20h⟩≡ (291c)
```
let document_id wwwr =
  match wwwr.www_link.h_method with
  | POST _  ->
        { document_url = wwwr.www_url; document_stamp = new_stamp()}
  | _ -> { document_url = wwwr.www_url; document_stamp = no_stamp}
```

⟨*module Document.DocumentIDSet* 20i⟩≡ (291c)
```
module DocumentIDSet =
  Set.Make(struct type t = id let compare = compare end)
```

3.2.2 Document cache

⟨*signature* Cache.add 20j⟩≡ (309l)
```
val add : Document.id -> Document.t -> unit
```

⟨*signature* Cache.find 20k⟩≡ (309l)
```
val find : Document.id -> Document.t
```

⟨*signature* Cache.finished 20l⟩≡ (309l)
```
val finished : Document.id -> unit
```

⟨*signature* Cache.touch 20m⟩≡ (309l)
```
val touch : Document.id -> unit
```

⟨*signature* Cache.kill 20n⟩≡ (309l)
```
val kill : Document.id -> unit
```

⟨*constant* Cache.memory 21a⟩≡ (314e)
```
let memory = ref ([] : (Document.id * entry) list)
```

⟨*type* Cache.entry 21b⟩≡ (314e)
```
(* A cache entry *)
type entry = {
  mutable cache_document : Document.t;

  mutable cache_pending : bool;
  cache_condition : Condition.t;

  mutable cache_lastused : float
      (* TODO still valid comment in 2026? *)
      (* old? cache_lastused is specified as max_int (0x3fffffff) when we don't
       * want the entry to be flushed. This will break around
       * Sat Jan 10, 2004 13:37 GMT on 32 bits machines
       *) (* JPF: it is now float! max_int -> max_float *)
  }
```

⟨*constant* Cache.max_lastused 21c⟩≡ (314e)
```
(* Corresponding Date = year 5138, we should be fine *)
let max_lastused = 100000000000.0
```

⟨*function* Cache.find 21d⟩≡ (314e)
```
(* Find a document*)
let find (did : Document.id) : Document.t =
  let entry = List.assoc did !memory in
  entry.cache_lastused <- Unix.time();
  if entry.cache_pending
  then Condition.wait entry.cache_condition;
  entry.cache_document
```

3.3 Protocols and document flow

⟨*signature* Protos.get 21e⟩≡ (317b)
```
val get: Url.protocol ->
  (< Cap.network > -> Www.request -> Document.continuation -> Www.aborter)
  *
  (Document.handle -> Document.data * Cache.cache_fill)
```

3.3.1 Protos.protos

⟨*constant* Protos.protos 21f⟩≡ (318a)
```
let protos : (Url.protocol,
              (< Cap.network > -> Www.request -> Document.continuation -> Www.aborter) *
              (Document.handle -> Document.data * Cache.cache_fill)) Hashtbl.t =
  Hashtbl_.create ()
```

⟨*constant* Protos.get 21g⟩≡ (318a)
```
let get = Hashtbl.find protos
```

3.3.2 Document continuation

⟨*type* Document.document_continuation 22a⟩≡ (291)
```
  type continuation = {
    document_process : handle -> unit;
      (* What to do one we have a dh on the real document *)

    document_finish :  bool -> unit
      (* What to do if a request does not yield a document. bool = ???? *)
  }
```

3.3.3 Document handle

⟨*type* Document.handle 22b⟩≡ (291)
```
  (* This is passed around by request continuations. It represents a handle
     on a connexion for retrieving a document *)
  type handle = {
    document_id : id;

    (* this should help to know what to do even if have not the data yet.
     * Those are response headers.
     *)
    mutable dh_headers : string list;
      (* HTTP headers of document, or faked ones *)

    document_feed : Feed.t;
      (* where to get the data *)

    ⟨Document.handle other fields 105b⟩
  }
```

3.4 HTTP

3.4.1 Requests

⟨*type* Messages.request_message 22c⟩≡ (300a)
```
  (* HTTP-Message *)
  type request = {
    request : request_line;
    request_headers : header list;
    request_body : string;
    ⟨Messages.request_message other fields 229d⟩
  }
```

⟨*type* Messages.request 22d⟩≡ (300a)
```
  (* Request-Line of a Request *)
  type request_line = {
    request_version: string; (* HTTP/1.0 *)
    request_method : string; (* GET, POST, etc... *)

    request_uri : string  (* the uri *)
  }
```

3.4.2 Responses

⟨*type* Messages.response_message 23a⟩≡ (300a)
```
  type response = {
    status : status_line;

    response_headers : header list;
    response_body : string;        (* responde body is *not* the document body *)
  }
```

⟨*type* Messages.status 23b⟩≡ (300a)
```
  (* Status-Line of a Response *)
  type status_line =  {
      status_version : string; (* HTTP/1.0 *)
      status_code : int;  (* http return codes *)
      status_message : string (* http return message *)
  }
```

3.4.3 Headers and content types

⟨*type* Messages.header 23c⟩≡ (300a)
```
  (* Other headers *)
  type header = string
```

⟨*type* Http_headers.media_type 23d⟩≡ (302 300g)
```
  (* type/sub, ex: text/html, images/gif, applications/postscript *)
  type media_type = string * string
```

⟨*type* Http_headers.media_parameter 23e⟩≡ (302 300g)
```
  (* ex: ?? *)
  type media_parameter = string * string
```

3.5 Viewers

⟨*type* Viewers.t 23f⟩≡ (340c 339c)
```
  (* Definition of an internal viewer *)
  type t =
      Http_headers.media_parameter list ->
      (Widget.widget -> context -> Document.handle -> display_info option)
```

3.5.1 Viewers.viewers

⟨*constant* Viewers.viewers 23g⟩≡ (340c)
```
  let viewers : (Http_headers.media_type, spec) Hashtbl.t =
    Hashtbl_.create ()
```

⟨*type* Viewers.spec 23h⟩≡ (340c)
```
  type spec =
    | Internal of t
    | External
    ⟨Viewers.spec other cases 120a⟩
```

3.5.2 `Viewers.context`

⟨*signature class* `Viewers.context` 24a⟩≡ (339c)
```
  (* The context given to a viewer *)
  class virtual context : (Document.id * vparams) -> object ('a)

    method base : Document.id

    ⟨Viewers.context hypertext methods signatures 24b⟩
    ⟨Viewers.context embedded methods signatures 163c⟩

    ⟨Viewers.context logging methods signatures 246d⟩
    ⟨Viewers.context other methods signatures 163d⟩
  end
```

TODO XXX ??? when is this called ? why need pass down in context to viewer?

⟨`Viewers.context` *hypertext methods signatures* 24b⟩≡ (24a) 24c▷
```
  method goto    : Hyper.link -> unit
  method gotonew : Hyper.link -> unit
  method save    : Hyper.link -> unit
```

⟨`Viewers.context` *hypertext methods signatures* 24c⟩+≡ (24a) ◁24b
```
  method invoke   : string -> Hyper.link -> unit
  method add_nav     : string * hyper_func -> unit
  method hyper_funs : (string * hyper_func) list
```

⟨*type* `Viewers.hyper_func` 24d⟩≡ (340c 339c)
```
  type hyper_func = {
    hyper_visible : bool;
    hyper_title : string;

    hyper_func : frame_targets -> Hyper.link -> unit
  }
```

⟨*class* `Viewers.context` 24e⟩≡ (340c)
```
  (* The context given to a viewer *)
  class virtual context ((did : Document.id),
                         (v : vparams)) =
   object (self : 'a)

    val base = did
    method base = base

    val viewer_params = v
    method params = viewer_params

    val mutable (*private*) funs = ([] : (string * hyper_func ) list)
    method hyper_funs = funs

    val targets = []

    method goto hlink    = self#invoke "goto" hlink
    method gotonew hlink = self#invoke "gotonew" hlink
    method save hlink    = self#invoke "save" hlink

    method invoke name hlink =
      try (List.assoc name funs).hyper_func targets hlink
      with Not_found -> ()

    method add_nav (fname, hf) =
      funs <- (fname, hf) :: funs

    ⟨method Viewers.context.for_embed 162a⟩
    ⟨method Viewers.context.in_embed 162b⟩

    method virtual log : string -> unit
  end
```

3.5.3 Viewers.display_info

⟨*signature class* Viewers.display_info 25a⟩≡ (339c)

```
  class  virtual display_info : (unit) -> object ('a)

    ⟨Viewers.display_info virtual methods signatures 25b⟩
    ⟨Viewers.display_info graphic cache methods signatures 237c⟩
  end
```

⟨Viewers.display_info *virtual methods signatures* 25b⟩≡ (340c 25a)

```
  method virtual di_title : string  (* some visible title *)

  (* the created widget containing the graphics *)
  method virtual di_widget : Widget.widget

  ⟨Viewers.display_info images virtual methods signatures 169d⟩
  ⟨Viewers.display_info embedded virtual methods signatures 164d⟩
  ⟨Viewers.display_info fragment virtual method signature 199c⟩

  ⟨Viewers.display_info lifecycle virtual methods signatures 203b⟩

  ⟨Viewers.display_info graphic cache virtual methods signatures 238a⟩
  ⟨Viewers.display_info other virtual methods signatures 192c⟩
```

3.6 Abstract syntax trees

3.6.1 HTML

⟨*type* Html.token 25c⟩≡ (295 293d)

```
  type token =
   | Doctype of string

   | OpenTag of tag
   | CloseTag of string

   | PCData of string
   | CData of string

   | Comment of string

   | EOF
```

⟨*type* Html.tag 25d⟩≡ (295 293d)

```
  type tag = {
    tag_name : string;
    attributes: attributes
  }
```

⟨*type* Html.attributes 25e⟩≡ (295 293d)

```
  type attributes = (attribute_name * attribute_value) list
```

⟨*type* Html.attribute_name 26a⟩≡ (295 293d)
```
  type attribute_name = string
```

⟨*type* Html.attribute_value 26b⟩≡ (295 293d)
```
  type attribute_value = string
```

3.6.2 CSS

3.6.3 Javascript

3.7 DOM

3.8 HTML display

3.8.1 Formatter

⟨*type* Htmlfmt.gattr 26c⟩≡ (374c)
```
  type gattr =
    ⟨Htmlfmt.gattr color cases 157b⟩
    ⟨Htmlfmt.gattr font cases 155c⟩
    ⟨Htmlfmt.gattr spacing cases 141c⟩
    ⟨Htmlfmt.gattr alignment cases 137e⟩
    ⟨Htmlfmt.gattr style cases 142e⟩
```

⟨*type* Htmlfmt.formatter 26d⟩≡ (374c)
```
  type formatter = {

    (* Text primitives of the device *)
    ⟨Htmlfmt.formatter primitives methods 138a⟩

    (* Graphical attributes *)
    ⟨Htmlfmt.formatter graphical attributes methods 139c⟩

    ⟨Htmlfmt.formatter other methods 26e⟩

    flush : unit -> unit;   (* Flush the device *)
  }
```

XXX

⟨Htmlfmt.formatter *other methods* 26e⟩≡ (26d)
```
  (* Predefined Images *)
  ⟨Htmlfmt.formatter predefined images methods 144b⟩
  (* Structure primitives *)
  ⟨Htmlfmt.formatter structure primitives methods 136b⟩
  (* Embedding primitives *)
  ⟨Htmlfmt.formatter embedding primitives methods 170a⟩
  (* Re-centering on a fragment *)
  ⟨Htmlfmt.formatter fragment method 199j⟩
```

⟨*constant* Html_disp.default_fo 26f⟩≡ (423b)
```
  (* This is the default formatter *)
  let default_fo = Htmlfmt.{
    (* Text primitives of the device *)
    new_paragraph   = (fun () -> ());
    close_paragraph = (fun () -> ());
    print_newline   = (fun _b -> ());
    print_verbatim  = (fun _s -> ());
    format_string   = (fun _s -> ());

    (* Graphical attributes *)
    push_attr       = (fun _l -> ());
    pop_attr        = (fun _l -> ());
    set_defaults    = (fun _s _l -> ());

    (* misc *)

    hr              = (fun _l _n _b -> ());
    bullet          = (fun _n -> ());


    isindex         = (fun _s _s' -> ());
    start_anchor    = (fun () -> ());
    end_anchor      = (fun _h -> ());
    add_mark        = (fun _ -> ());

    create_embedded = (fun _a _w _h -> assert false);

    see_frag        = (fun _ -> ());

    flush           = (fun () -> ());
  }
```

3.8.2 Interpreter

⟨*signature class* Html_disp.machine 27a⟩≡ (423a)

```
  class  virtual machine : (unit) -> object
    ⟨Html_disp.machine virtual fields signatures 27b⟩
  end
```

⟨Html_disp.machine *virtual fields signatures* 27b⟩≡ (28 27a)

```
  (* context *)
  method virtual ctx : Viewers.context

  (* input *)
  method virtual send : Html.token -> unit
  ⟨Html_disp.machine html input other methods 128c⟩

  (* semantic *)
  method virtual add_tag:
    string ->
    (* open handler *)  (Htmlfmt.formatter -> Html.tag -> unit) ->
    (* close handler *) (Htmlfmt.formatter -> unit) ->
    unit
  method virtual remove_tag : string -> unit
  ⟨Html_disp.machine tags methods 126g⟩
  ⟨Html_disp.machine action stack methods 127a⟩

  (* backend *)
  method virtual formatter : Htmlfmt.formatter
  ⟨Html_disp.machine formatter stack methods 127f⟩
  ⟨Html_disp.machine formatter misc methods 127h⟩

  (* special tags *)
  ⟨Html_disp.machine embedded fields 163a⟩
  ⟨Html_disp.machine image methods 169a⟩
  ⟨Html_disp.machine fragment method 199g⟩
  (* misc *)
  ⟨Html_disp.machine i18 methods 226k⟩
  ⟨Html_disp.machine other fields 126a⟩
```

⟨*class* `Html_disp.machine` 28⟩≡ (423b)

```
  class  virtual machine (_unit : unit) =
   object
     ⟨Html_disp.machine virtual fields signatures 27b⟩
  end
```

3.9 Summary

Chapter 4

main()

⟨*type* Main.caps 29a⟩≡ (463b)
```
(* Need:
 *  - Cap.network: obviously, this is a Web browser
 *  - TODO: Cap.exec for mmmc, convert (ImageMagic jpeg converter), metamail
 *  - open_in: for file://, for ??
 *)
type caps = <
   Cap.open_in;
   Cap.network
 >
```

⟨Main.main() *locals* 29b⟩≡ (29c) 30e ▷
```
let init_urls = ref [] in
```

⟨*function* Main.main 29c⟩≡ (463b)
```
let main (caps : < caps; Cap.stdout; Cap.stderr; .. >)
         (argv : string array) : Exit.t =
  ⟨Main.main() tk backends setup 30b⟩

  (* As always, we must parse argument first, using references... *)
  ⟨Main.main() locals 29b⟩
  let level = ref (Some Logs.Warning) in

  let options = ([
   ⟨Main.main() command line options 31a⟩
  ] @ Logs_.cli_flags level) |> Arg.align
  in
  Arg_.parse_argv caps argv options
    (fun s -> init_urls := s :: !init_urls)
    usage_str
  ;
  Logs_.setup !level ();
  Logs.info (fun m -> m "ran as %s from %s" argv.(0) (Sys.getcwd()));
  ⟨Main.main() signal handling 240⟩
  ⟨Main.main() initialisation 30c⟩

  let url_opt =
    match !init_urls with
    | []     -> None
    | x::_l -> Some x
  in
  let user_preferences_file : Fpath.t =
    ⟨Main.main() user preferences file 208e⟩
  in
  (* Start the initial navigator *)
```

```
    Mmm.initial_navigator caps user_preferences_file url_opt;

    safe_loop();
    ⟨Main.main() after event loop, if debug mode 243g⟩
    Exit.OK
```

⟨*signature* Mmm.initial_navigator 30a⟩≡ (459)

```
  val initial_navigator : < Cap.network; .. > ->
    Fpath.t (* preference file *) -> string (* url *) option -> unit
```

XXX

⟨Main.main() *tk backends setup* 30b⟩≡ (29c)

```
  Error.default                       := new Tk_error.t Widget.default_toplevel;
  Condition.backend                   := Tk_condition.backend ();
  Timer_.add_ref                      := (fun a b -> Timer.add a b |> ignore);
  Timer_.set_ref                      := Timer.set;
  Low.update_idletasks_backend        := Tk.update_idletasks;
  Fileevent_.add_fileinput_ref        := Fileevent.add_fileinput;
  Fileevent_.remove_fileinput_ref     := Fileevent.remove_fileinput;
  Fileevent_.add_fileoutput_ref       := Fileevent.add_fileoutput;
  Fileevent_.remove_fileoutput_ref    := Fileevent.remove_fileoutput;
  Document.add_log_backend            := Tk_document.add_log;
  Maps.broadcast_backend              := Frx_synth.broadcast;
  Auth.open_passwd_ref                := Frx_req.open_passwd;
  Auth.edit_backend                   := Tk_auth.edit;
  Mailto.internal_backend             := Tk_mailto.internal;
```

4.1 Initialisations

⟨Main.main() *initialisation* 30c⟩≡ (29c)

```
  ⟨Main.main() tk initialisation 30d⟩
  ⟨Main.main() resource initialisation 31d⟩
  ⟨Main.main() tk libs initialisation 31c⟩
  ⟨Main.main() local initialisation 31f⟩
  ⟨Main.main() suffix initialisation 117d⟩
  ⟨Main.main() misc initialisation 122b⟩
  ⟨Main.main() html entities initialisation 67d⟩
  ⟨Main.main() applet system initialisation 176g⟩
  ⟨Main.main() mmm server initialisation 219e⟩
```

4.1.1 Graphics initialisation

⟨Main.main() *tk initialisation* 30d⟩≡ (30c) 209a▷

```
  let top = Tk.openTkDisplayClass !display "mmm" in
  Wm.withdraw top;
  (* Load tkimg if available so Tk can handle JPEG (and TIFF) natively.
     On Debian/Ubuntu install: sudo apt-get install libtk-img *)
  (try Protocol.tkCommand [|Protocol.TkToken "package";
                            Protocol.TkToken "require";
                            Protocol.TkToken "Img"|]
   with Protocol.TkError _ ->
     Logs.info (fun m -> m "tkimg not available; JPEG will use %s fallback"
       !Img.ImageData.jpeg_converter));
```

⟨Main.main() *locals* 30e⟩+≡ (29c) ◁29b 117b▷

```
  let display = ref (try Sys.getenv("DISPLAY") with Not_found -> "") in
```

⟨Main.main() *command line options* 31a⟩≡ (29c) 31b ▷
```
  "-d", Arg.String (fun s -> display := s),
  " <foo:0> Display";
```

⟨Main.main() *command line options* 31b⟩+≡ (29c) ◁ 31a 117c ▷
```
  "-display", Arg.String (fun s -> display := s),
  " <foo:0> Display";
```

⟨Main.main() *tk libs initialisation* 31c⟩≡ (30c)
```
  (* Initialisations in frx library : kbd navigation, search
   * No prerequisite except Tk *)
  Frx_text.init ();
  (* Initialisations in jpf's balloon library *)
  Balloon.init ();
  (* Initialisations in jpf's GIF ANIMATION library *)
  (* TODO: Tkaniminit.f (); linking problem *)
```

4.1.2 Resources initialisation

⟨Main.main() *resource initialisation* 31d⟩≡ (30c) 208j ▷
```
  (* Default values for navigator window
   * old: was 640x480, but does not seem to fully work, xwininfo returns
   * different values of the one specified below.
   *)
  Resource.add "*MMM.Width" "2024" Tk.WidgetDefault;
  Resource.add "*MMM.Height" "1768" Tk.WidgetDefault;

  (* Resources *)
  let site_resfile =
    localize (Fpath.v (Filename.dirname argv.(0)) / "data/MMM.ad") in
  (* Site specific resource file usually in INSTALLDIR=/usr/local/lib/mmm *)
  if Sys.file_exists !!site_resfile
  then  begin
      Logs.info (fun m -> m "loading resource startup file %s" !!site_resfile);
      Tkresource.readfile !!site_resfile Tk.StartupFile
  end;
```

⟨*function* Main.localize 31e⟩≡ (463b)
```
  let localize (file : Fpath.t) : Fpath.t =
    let localized = spf "%s.%s" !!file !I18n.language in
    if Sys.file_exists localized
    then Fpath.v localized
    else file
```

4.1.3 Local initialisation

⟨Main.main() *local initialisation* 31f⟩≡ (30c)
```
  (* Local initialisations *)
  Low.init();                         (* start regular tasks *)
  Cache.init();                       (* builtin document *)
  Auth.init();                        (* start expiration timer *)
  Debug.init();                       (* debugging RPC *)
```

4.1.4 Initial fake URL and HTML document

⟨*signature* Cache.init 31g⟩≡ (309l)
```
  val init : unit -> unit
```

⟨*function* Cache.init 32a⟩≡ (314e)
```
  let init () =
    let initurl = Lexurl.make (Version.initurl (Lang.lang ())) in

    let b = Ebuffer.create 128 in
    Ebuffer.output_string b (Version.inithtml (Lang.lang ()));

    let docid = { document_url = initurl; document_stamp = Document.no_stamp; } in
    let doc = {
      document_headers = ["Content-Type: text/html"];
      document_data = MemoryData b;
      document_address = initurl;
    } in
    let docentry = {
      cache_document = doc;
      cache_pending = false;
      cache_condition = Condition.create();
      cache_lastused = max_lastused;
    }
    in

    memory := [docid, docentry];
    current := 1
```

⟨*signature* Version.initurl 32b⟩≡ (285e)
```
  val initurl : string -> string (* fake initial url *)
```

⟨*function* Version.initurl 32c⟩≡ (286a)
```
  (* MUST BE NORMALIZED *)
  let initurl = function
    | "iso8859" ->
        Printf.sprintf "http://pauillac.inria.fr/mmm/v%d/about.html" number
    | s -> failwith (Printf.sprintf "language not supported: %s" s)
```

⟨*signature* Version.html 32d⟩≡ (285e)
```
  val inithtml : string -> string    (* fake initial document *)
```

⟨*function* Version.html 32e⟩≡ (286a)
```
  let inithtml = function
    | "iso8859" ->
  "<HTML>
    <HEAD><TITLE>MMM 0." ^ version_number ^ "</TITLE></HEAD>
  <BODY>
  <H1> The MMM navigator Version 0." ^ version_number ^ "</H1>
  <H2 ALIGN=CENTER> Written by Fran\231ois Rouaix </H2>
  <H2 ALIGN=CENTER> Contributions by Jun P. Furuse and Jacques Garrigue</H2>
  <H3 ALIGN=CENTER> Port to O'Caml V3.0 by Jun P. Furuse and Pierre Weis</H3>
  <H2 ALIGN=CENTER> \169 Copyright INRIA </H2>

  <H4 ALIGN=CENTER> Using Objective Caml \169 Copyright INRIA </H4>
  <H4 ALIGN=CENTER> And Tcl8.0/Tk8.0 (John Ousterhout and al.)<BR>
   \169 Copyright The Regents of the University of California<BR>
   and Sun Microsystems, Inc </H4>
  <BLOCKQUOTE>
  Please note that the software is a product currently being developed.
  INRIA shall not be responsible in any way concerning conformity, and in
  particular shall not be liable should the software not comply with the
  requirements of the user, INRIA not being obliged to repair any
  possible direct or indirect damage.
  </BLOCKQUOTE>
```

```
<P>
The MMM home page is
<A HREF='http://pauillac.inria.fr/mmm/'>here</A>,
and there is also some
<A HREF='http://pauillac.inria.fr/mmm/doc.html'>documentation</A>
and
<A HREF='http://pauillac.inria.fr/mmm/releases.html'>release notes</A>.
<BR>
Join the author by clicking
<A HREF='mailto:Francois.Rouaix@inria.fr'>here.</A>
<P>
<BLOCKQUOTE>
This document is included in your browser. Click on <TT>Reload</TT> to
get an updated copy.
</BLOCKQUOTE>
</BODY>
</HTML>
"
  |  s -> failwith (Printf.sprintf "language %s not supported here" s)
```

4.2 Mmm.initial_navigator()

⟨*function* Mmm.initial_navigator 33a⟩≡ (460)

```
(* main -> <> -> navigator *)
let initial_navigator (caps : < Cap.network; ..>)
    (preffile : Fpath.t) (init_url : string option) : unit =
  ⟨Mmm.initial_navigator() set preferences 207h⟩
  ⟨Mmm.initial_navigator() set initial page based on init_url 33d⟩
  main_navigator :=
     navigator caps true
       (match !initial_page with Some u -> u | None -> assert false)
```

⟨*signature* Mmm.navigator 33b⟩≡ (459)

```
val navigator : < Cap.network; .. > ->
  bool (* is_main_window *) -> Url.t -> Nav.t option
```

4.2.1 Initial URL

⟨*constant* Mmm.initial_page 33c⟩≡ (460)

```
let initial_page : Url.t option ref = ref None
```

⟨Mmm.initial_navigator() *set initial page based on* init_url 33d⟩≡ (33a)

```
initial_page := Some (
   match init_url with
   | None -> Lexurl.make !Mmmprefs.home
   | Some x ->
       (try Lexurl.make x
        with _ -> (* If fails, try to use file: *)
          ⟨Mmm.initial_navigator() if cannot parse init_url 33e⟩
       )
);
```

⟨Mmm.initial_navigator() *if cannot parse* init_url 33e⟩≡ (33d)

```
let path =
  if x.[0] = '/'
  then x
  else Filename.concat (Unix.getcwd ()) x
in
Lexurl.make ("file://localhost" ^ path)
```

4.2.2 Mmm.navigator() and Nav.t

```
⟨function Mmm.navigator 34a⟩≡                                                    (460)
  (* (main -> initial_navigator) | navigator -> <> -> Nav.absolutegoto *)
  let rec navigator (caps: < Cap.network; ..>)
          (is_main_window: bool) (initial_url : Url.t) : Nav.t option =
    ⟨Mmm.navigator() new navigator hook 206h⟩

    (* The first navigator is named, so we can put special information in
     * window manager configurations, such as sticky (??)
     *)
    let top =
      if is_main_window
      then Toplevel.create_named Widget.default_toplevel "mmm" [Class "MMM"]
      else Toplevel.create       Widget.default_toplevel       [Class "MMM"]
    in
    Wm.title_set top (s_ "MMM Browser");
    ⟨Mmm.navigator() setup top packing 41b⟩

    ⟨Mmm.navigator() locals 38d⟩

    (* protect all the other initialisations *)
    try
      (* The frame in which a viewer might want to display *)
      let viewer_frame = Frame.create_named top "viewer" [] in

      ⟨Mmm.navigator() locals before nav setting 34c⟩
      let nav = Nav.{
        ⟨Mmm.navigator() set nav fields 35c⟩
      }
      in
      ⟨Mmm.navigator() nested functions 35a⟩
      ⟨Mmm.navigator() keyboard shortcuts setting 48e⟩
      ⟨Mmm.navigator() widgets setting 41a⟩

      Nav.absolutegoto caps nav (Url.string_of initial_url);
      Some nav

    with e ->
      Error.f (s_ "Can't view initial document: %s\n%s"
                          (Url.string_of initial_url)
                          (Printexc.to_string e));
      if !navigators = 1 then begin
        Tk.destroy Widget.default_toplevel;
        raise e
      end
      ⟨Mmm.navigator() exn handler, when multiple navigators 206j⟩
```

```
⟨signature Nav.absolutegoto 34b⟩≡                                               (450f)
  val absolutegoto : < Cap.network; ..> ->
    t -> string (* url *) -> unit
```

```
⟨Mmm.navigator() locals before nav setting 34c⟩≡                                 (34a)
  ⟨local Mmm.navigator.hist 42j⟩
  ⟨local function Mmm.navigator.show_current 38a⟩
  ⟨local function Mmm.navigator.add_hist 42h⟩
  ⟨local object Mmm.navigator.error 48c⟩
  ⟨local Mmm.navigator.loggingv 47h⟩
  ⟨local Mmm.navigator.actives 202d⟩
```

⟨Mmm.navigator() *nested functions* 35a⟩≡ (34a) 207f▷
```
  (* The navigation functions *)

  ⟨function Mmm.navigator.back 43b⟩
  ⟨function Mmm.navigator.forward 43d⟩
  ⟨function Mmm.navigator.reload 203h⟩
  ⟨function Mmm.navigator.update 204a⟩

  (* A bunch of other functions *)

  ⟨function Mmm.navigator.abort 202c⟩
  ⟨function Mmm.navigator.open_sel 44e⟩
  ⟨function Mmm.navigator.open_file 45a⟩
  ⟨function Mmm.navigator.save 45b⟩
  ⟨function Mmm.navigator.print 45d⟩
  ⟨function Mmm.navigator.close 45e⟩
  ⟨function Mmm.navigator.really_quit 45f⟩
  ⟨function Mmm.navigator.gohome 43f⟩
  ⟨function Mmm.navigator.redisplay 204d⟩
  ⟨function Mmm.navigator.add_to_hotlist 206b⟩
  ⟨function Mmm.navigator.load_images 169c⟩
  ⟨function Mmm.navigator.view_source 192b⟩
```

⟨*type* Nav.t 35b⟩≡ (453 450f)
```
  type t = {
    nav_viewer_frame : Widget.widget;

    (* Nav.absolutegoto -> request -> process_viewer -> <> *)
    nav_show_current: Viewers.display_info -> string option (* frag *) -> unit;

    ⟨Nav.t manage history methods 42f⟩
    ⟨Nav.t manage active connections methods 202e⟩

    ⟨Nav.t graphic cache related methods 238d⟩
    ⟨Nav.t error methods 48b⟩
    ⟨Nav.t logging method 47i⟩

    ⟨Nav.t other fields 206k⟩
   }
```

⟨Mmm.navigator() *set nav fields* 35c⟩≡ (34a) 37d▷
```
  nav_viewer_frame = viewer_frame;
```

4.2.3 Nav.absolutegoto()

⟨*function* Nav.absolutegoto 35d⟩≡ (453)
```
  (* Used outside an hyperlink *)
  (* main -> Mmm.initial_navigator -> Mmm.navigator -> <> -> follow_link ->
   *  request -> Retrieve.f -> Http.req (via protos) ->
   *   process_viewer (via cont) ->
   *    Viewer.f (as di); nav.nav_show_current di -> Mmm.show_current
   * 'open URL:' entry | Mmm.navigator.open_sel | ... -> <>
   *)
  let absolutegoto (caps : < Cap.network; ..>) (nav : t) (uri : string) =
    follow_link caps nav (Hyper.default_link uri)
```

⟨*signature* Nav.follow_link 35e⟩≡ (450f)
```
  val follow_link : < Cap.network; ..> ->
    t -> Hyper.link -> unit
```

⟨*function* Nav.follow_link 36a⟩≡ (453)
```
  let follow_link (caps : < Cap.network; ..>)
       (nav : t) (lk : Hyper.link) : unit =
    lk |> request caps nav (fun (nav : t) (wr : Www.request) (dh : Document.handle) ->
        process_viewer true (make_ctx caps) nav wr dh
      )
      ⟨Nav.follow_link extra arguments to Nav.request 40a⟩
```

⟨*signature* Nav.request 36b⟩≡ (450f)
```
  val request : < Cap.network; ..> ->
    t -> (t -> Www.request -> Document.handle -> unit) ->
    ⟨Nav.request signature, extra arguments 40c⟩
    Hyper.link ->
    unit
```

⟨*signature* Nav.make_ctx 36c⟩≡ (450f)
```
  val make_ctx : < Cap.network; ..> ->
    t -> Document.id -> Viewers.context
```

4.2.4 Nav.request()

⟨*function* Nav.request 36d⟩≡ (453)
```
  (* [request nav usecache wrapwr process specific] produces a function that
     takes an hyperlink, and apply the given behavior to it.
     [usecache] : do we look in the cache to see if we have it already
     [process nav wr dh] : what to to with the retrieved document
     [specific nav did wr] : some specific behavior, checked before we
       look in the cache. Must either raise Not_found or process completely
       the link
     [wrapwr wr] : returns a modified wr
   *)
  let request (caps : < Cap.network; ..>) (nav : t)
       process (usecache, wrapwr, specific) =
    fun (lk : Hyper.link) ->

    ⟨function Nav.request.retrieve_and_handle 37b⟩
    ⟨function Nav.request.handle_wr 37a⟩
    ⟨function Nav.request.handle_link 36e⟩
    handle_link lk
```

⟨*function* Nav.request.handle_link 36e⟩≡ (36d)
```
  and handle_link (h : Hyper.link) : unit =
    try (* Convert the link into a request *)
      let wr = Plink.make h in
      wr.www_error <- nav.nav_error;
      wr |> wrapwr |> handle_wr
    with
    | Hyper.Invalid_link _msg ->
        nav.nav_error#f (s_ "Invalid link")
    | Www.Invalid_request (wr, msg) ->
        nav.nav_error#f (s_ "Invalid request %s\n%s"(Url.string_of wr.www_url)msg)
  in
```

⟨*signature* Plink.make 36f⟩≡ (434b)
```
  val make : Hyper.link -> Www.request
      (* [make hlink] is an error correcting version of Www.make
         For invalid links, a dialog box is displayed and offers
         edition facilities
       *)
```

⟨*function* Nav.request.handle_wr 37a⟩≡ (36d)
```
  (* Wrapper to deal with general/specific cache *)
  and handle_wr (wr : Www.request) : unit =
    try
      match wr.www_url.protocol with
      ⟨Nav.request.handle_wr() match protocol special cases 213c⟩
      | _ ->
         if (not usecache) || dont_check_cache wr
         then retrieve_and_handle wr
         else
           ⟨Nav.request.handle_wr() if use cache 236a⟩
    with Duplicate url ->
      wr.www_error#f (s_ "The document %s\nis currently being retrieved for some other purpose.\nMMM cannot proce
```

⟨*function* Nav.request.retrieve_and_handle 37b⟩≡ (36d)
```
  (* Normally execute the request and process its answer (dh) *)
  (* handle_link -> handle_wr -> <> *)
  let rec retrieve_and_handle (wr : Www.request) =
    let cont =
      Document.{ document_process = (fun dh ->
          process nav wr dh;
          nav.nav_rem_active wr.www_url
        );
        document_finish = (fun _ ->
          nav.nav_rem_active wr.www_url
        );
      }
    in
    (* ! Retrieve ! *)
    match Retrieve.f caps wr handle_link cont with
    | Retrieve.Started aborter ->
        nav.nav_add_active wr.www_url aborter
    | Retrieve.InUse ->
        raise (Duplicate wr.www_url)
```

4.2.5 Nav.process_viewer()

⟨*function* Nav.process_viewer 37c⟩≡ (453)
```
  (* Specific handling of "view" requests
   * Nav.absolutegoto -> Nav.follow_link -> Nav.request <> -> <> (via process)
   *  -> Viewers.f -> Htmlw.viewer | Plan.viewer
  *)
  let process_viewer (addhist : bool) make_ctx =
   fun nav _wr (dh : Document.handle) ->
    let ctx = make_ctx nav dh.document_id in
    (* ! Viewers ! *)
    match Viewers.f nav.nav_viewer_frame ctx dh with
    | None -> () (* external viewer *)
    | Some di ->
        ⟨Nav.process_viewer() add in cache and history the document 42e⟩
        nav.nav_show_current di dh.document_fragment
```

4.2.6 Mmm.show_current()

⟨Mmm.navigator() *set nav fields* 37d⟩+≡ (34a) ◁35c 42g▷
```
  nav_show_current = show_current;
```

⟨*local function* `Mmm.navigator.show_current` 38a⟩≡ (34c)
```
  (* Change view, independently of history manip *)
  (* Nav.absolutegoto -> Nav.request -> Nav.process_viewer -> <>
   *   (as Nav.nav_show_current)
   *)
  let show_current (di : Viewers.display_info) (frag : string option) =
    ⟨Mmm.navigator.show_current() start hook 237b⟩
    ⟨Mmm.navigator.show_current() possibly undisplay previous displayinfo 38e⟩
    display di;
    ⟨Mmm.navigator.show_current() goto fragment 199b⟩
    ⟨Mmm.navigator.show_current() end hook 38g⟩
  in
```

⟨*function* `Mmm.display` 38b⟩≡ (460)
```
  let display (di : Viewers.display_info) =
    if Winfo.exists di#di_widget
    then pack [di#di_widget][Fill Fill_Both; Expand true]
    else Error.f "fatal error: window was destroyed";

    ⟨Mmm.display() adjust title toplevel 38c⟩
```

⟨`Mmm.display()` *adjust title toplevel* 38c⟩≡ (38b)
```
  let tl = Winfo.toplevel di#di_widget in
  let title = s_ "MMM Browser@%s" di#di_title in
  if Widget.known_class tl = "toplevel"
  then begin
    Wm.title_set tl title;
    Wm.iconname_set tl title
  end
```

⟨`Mmm.navigator()` *locals* 38d⟩≡ (34a) 42c ▷
```
  let current_di : Viewers.display_info option ref = ref None in
```

⟨`Mmm.navigator.show_current()` *possibly undisplay previous displayinfo* 38e⟩≡ (38a)
```
  (match !current_di with
  | Some olddi when olddi != di -> undisplay olddi
  | _ -> ()
  );
  current_di := Some di;
```

⟨*function* `Mmm.undisplay` 38f⟩≡ (460)
```
  let undisplay (di : Viewers.display_info) =
    if Winfo.exists di#di_widget
    then Pack.forget [di#di_widget]
```

XXX

⟨`Mmm.navigator.show_current()` *end hook* 38g⟩≡ (38a)
```
  (* Bof *)
  Textvariable.set entryv (Url.string_of hist.h_current.h_did.document_url)
```

4.2.7 `make_ctx()`

⟨*function* `Nav.make_ctx` 38h⟩≡ (453)
```
  let make_ctx (caps : < Cap.network; ..>)
       (nav : t) (did : Document.id) : Viewers.context =
    ((new stdctx caps (did, nav))#init :> Viewers.context)
```

⟨*class* Nav.stdctx 39⟩≡ (453)

```
  (* WARNING: we take copies of these objects, so "self" must *not* be
   * captured in a closure (it would always point to the old object).
   * A new object is created for each new top viewer (follow_link).
   * AND for each frame_goto operation.
   *)
  class stdctx (caps : < Cap.network; ..>) (did, nav) =
   object (self)
    inherit Viewers.context (did, []) as super
    (* val did = did *)
    (* val nav = nav *)

    method log = nav.nav_log
    method init =

      (* a new context for a toplevel window *)
      let make_ctx (caps : < Cap.network; ..>) nav did =
        ((new stdctx caps (did, nav))#init :> Viewers.context)
      in

      ⟨nested function Nav.stdctx.init.make_embed 162c⟩

      (* by default, use the cache, don't touch the request *)
      let follow_link (caps : < Cap.network; ..>) _ =
        request caps nav (process_viewer true (make_ctx caps))
          (true, id_wr, specific_viewer true)

      and save_link (caps : < Cap.network; ..>) _ =
        request caps nav (process_save None)
           (true, id_wr, nothing_specific)

      and copy_link _ =
        copy_link nav

      and head_link (caps : < Cap.network; ..>) =
        let f = request caps nav process_head (true, id_wr, nothing_specific) in
        (fun _ hlink -> f (make_head hlink))

      and new_link _ = nav.nav_new

      in

      ⟨nested function Nav.stdctx.init.frame_goto 162d⟩

      !user_navigation |> List.iter super#add_nav;

      ["copy", copy_link, s_ "Copy this Link to clipboard";
       "head", head_link caps, s_ "Headers of document";
       "save", save_link caps, s_ "Save this Link";
       "gotonew", new_link, s_ "New window with this Link";
       "goto", frame_goto caps, s_ "Open this Link";
      ] |> List.iter (fun (name, f, txt) ->
          self#add_nav (name, { hyper_visible = true;
                                hyper_func = f;
                                hyper_title = txt })
      );
      self

  end
```

4.2.8 XXX

⟨Nav.follow_link *extra arguments to Nav.request* 40a⟩≡ (36a)
```
  (true, id_wr, specific_viewer true)
```

⟨*function* Nav.id_wr 40b⟩≡ (453)
```
  let id_wr wr = wr
```

⟨Nav.request *signature, extra arguments* 40c⟩≡ (36b)
```
  ( bool *
    (Www.request -> Www.request) *
    (t -> Document.id -> Www.request -> unit)
  ) ->
```

4.3 The event loop

⟨*function* Main.safe_loop 40d⟩≡ (463b)
```
  let rec safe_loop () =
    try
      Printexc.print Tk.mainLoop () (* prints and reraises *)
    with
    | Out_of_memory -> raise Out_of_memory
    | Sys.Break -> raise Sys.Break
    | Stack_overflow -> raise Stack_overflow
    | _e ->
        flush Stdlib.stderr;
        safe_loop()
```

Chapter 5

Navigator Interface

5.1 Layout

⟨Mmm.navigator() *widgets setting* 41a⟩≡ (34a)

```
  (* Invariable part (the rest being the di stuff)
     hgroup: blah and tachymeter
   *)
  let hgroup = Frame.create_named top "hgroup" [] in
  let vgroup = Frame.create_named hgroup "vgroup" [] in (* Menus, open entry *)

  (* Menus *)
  let mbar = Frame.create_named vgroup "menubar" [] in
  ⟨Mmm.navigator() setup menu 44a⟩
  ⟨Mmm.navigator() setup open url entry 42d⟩

  (* Navigation buttons *)
  let buttons = Frame.create_named vgroup "buttons" [] in
  ⟨Mmm.navigator() navigation buttons 43a⟩

  ⟨Mmm.navigator() packing part one 41c⟩
  (* Initial window only *)
  if is_main_window then begin
    ⟨Mmm.navigator() set geometry if specified 208i⟩
    ⟨Mmm.navigator() set tachymeter 197a⟩
  end;
  ⟨Mmm.navigator() packing part two 42a⟩

  ⟨Mmm.navigator() handling destroy event 42b⟩
  Tkwait.visibility hgroup;

  ⟨Mmm.navigator() call update_vhistory 46c⟩
  ⟨Mmm.navigator() call touch_current to not swap displayed documents 237a⟩
```

⟨Mmm.navigator() *setup top packing* 41b⟩≡ (34a)

```
  (* the size of the navigator MUST NOT depend on what is displayed inside *)
  (* Instead, we rely on defaults for class MMM, *MMM.Width, *MMM.Height   *)
  Pack.propagate_set top false;
```

⟨Mmm.navigator() *packing part one* 41c⟩≡ (41a)

```
  pack [mbar][Anchor NW; Side Side_Top; Fill Fill_X];
  pack [backb;homeb;forwardb;reloadb;abortb; loggingb]
         [Side Side_Left; Fill Fill_X];
  pack [entry][Fill Fill_X; Expand true; Side Side_Bottom; Anchor SW];
  pack [buttons][Fill Fill_X];
```

⟨Mmm.navigator() *packing part two* 42a⟩≡ (41a)
```
  (* Pack last to avoid lossage when resizing *)
  pack [vgroup][Fill Fill_X; Expand true; Side Side_Left];
  pack [hgroup][Fill Fill_X];
  pack [viewer_frame][Fill Fill_Both; Expand true];
```

⟨Mmm.navigator() *handling destroy event* 42b⟩≡ (41a)
```
  (* We receive this event for each children destroyed because we are
     a toplevel *)
  bind top [[], Destroy] (BindSet ([Ev_Widget], (fun ei ->
    if ei.ev_Widget = top then begin
      ⟨Mmm.navigator() destroy navigator hook 206i⟩
      (* we were destroyed by wm *)
      if !navigators = 0 && Winfo.exists Widget.default_toplevel
      then Tk.destroy Widget.default_toplevel
    end
  )));
```

5.2 Address bar

⟨Mmm.navigator() *locals* 42c⟩+≡ (34a) ◁38d 42i▷
```
  let entryv = Textvariable.create_temporary top in
```

⟨Mmm.navigator() *setup open url entry* 42d⟩≡ (41a)
```
  (* URL display and edit *)
  let entry,e =
    Frx_entry.new_label_entry vgroup (s_ "Open URL:")
      (fun url -> Nav.absolutegoto caps nav url)
  in
  Entry.configure e [TextVariable entryv; TextWidth 40];
```

5.3 Canvas

5.4 History

⟨Nav.process_viewer() *add in cache and history the document* 42e⟩≡ (37c) 237g▷
```
  if addhist
  then nav.nav_add_hist dh.document_id dh.document_fragment;
```

⟨Nav.t *manage history methods* 42f⟩≡ (35b)
```
  nav_add_hist : Document.id -> string (* fragment *) option -> unit;
```

⟨Mmm.navigator() *set nav fields* 42g⟩+≡ (34a) ◁37d 48a▷
```
  nav_add_hist = add_hist;
```

⟨*local function* Mmm.navigator.add_hist 42h⟩≡ (34c)
```
  let add_hist (did : Document.id) (frag : string option) =
    History.add hist did frag;
    !update_vhistory ()
  in
```

⟨Mmm.navigator() *locals* 42i⟩+≡ (34a) ◁42c
```
  let update_vhistory = ref (fun () -> ()) (* duh *) in
```

⟨*local* Mmm.navigator.hist 42j⟩≡ (34c)
```
  let hist = History.create
    { document_url = initial_url; document_stamp = Document.no_stamp } in
```

5.4.1 Back button

⟨Mmm.navigator() *navigation buttons* 43a⟩≡ (41a) 43c ▷
```
  let backb = Button.create_named buttons
    "back" [Text (s_ "Back"); Command back ] in
```

⟨*function* Mmm.navigator.back 43b⟩≡ (35a)
```
  (*  The cache may have been cleared, so the document may be lost.
   *  historygoto implements the proper logic for this, taking care
   *  of non-unique documents.
   *)
  let back () =
    match History.back hist with
    | None -> ()
    | Some (did, frag) ->
        if not (Nav.historygoto caps nav did frag true)
        then History.forward hist |> ignore
  in
```

5.4.2 Forward button

⟨Mmm.navigator() *navigation buttons* 43c⟩+≡ (41a) ◁ 43a 43e ▷
```
  let forwardb = Button.create_named buttons
    "forward" [Text (s_ "Forward"); Command forward] in
```

⟨*function* Mmm.navigator.forward 43d⟩≡ (35a)
```
  let forward () =
    match History.forward hist with
    | None -> ()
    | Some (did, frag) ->
        if not (Nav.historygoto caps nav did frag true)
        then History.back hist |> ignore
  in
```

5.5 Home

⟨Mmm.navigator() *navigation buttons* 43e⟩+≡ (41a) ◁ 43c 47g ▷
```
  let homeb = Button.create_named buttons "home"
    [ Text (s_ "Home"); Command gohome] in
```

⟨*function* Mmm.navigator.gohome 43f⟩≡ (35a)
```
  let gohome () =
    Nav.absolutegoto caps nav !Mmmprefs.home
  in
```

⟨*signature* Mmmprefs.home 43g⟩≡ (454c)
```
  val home : string ref
```

⟨*constant* Mmmprefs.home 43h⟩≡ (458g)
```
  (* There is no right place for this *)
  let home = ref ""
```

TODO: so why go to mmm homepage by default??? special magic somewhere?

5.6 Menus

⟨Mmm.navigator() *setup menu* 44a⟩≡ (41a)
```
  ⟨function Mmm.navigator.configure_menu_elements 49a⟩

  ⟨Mmm.navigator() MMM menu 44b⟩
  ⟨Mmm.navigator() Navigation menu 46a⟩
  ⟨Mmm.navigator() History menu 46b⟩
  ⟨Mmm.navigator() Document menu 46d⟩
  ⟨Mmm.navigator() Other menu 47a⟩
  ⟨Mmm.navigator() Help menu 47b⟩
  ⟨Mmm.navigator() User menu 187f⟩

  pack [mmm; navb; docb; othersb][Side Side_Left];
  pack [helpb; userb] [Side Side_Right];
```

5.6.1 MMM menu

⟨Mmm.navigator() *MMM menu* 44b⟩≡ (44a)
```
  (* MMM menu *)
  let mmm = Menubutton.create_named mbar "mmm" [Text (s_ "MMM")] in
  let mmmm = Menu.create_named mmm "menu" [] in
  Menubutton.configure mmm [Menu mmmm];

  configure_menu_elements mmmm [
    [Label (s_ "About")             ; Command About.f];
    [];
    [Label (s_ "New Window")        ; Command new_window];
    [Label (s_ "Open Selection")    ; Command open_sel];
    [Label (s_ "Open File...")      ; Command open_file];
    [Label (s_ "Save document...") ; Command save];
    [Label (s_ "Print document")    ; Command print];
    [Label (s_ "Preferences...")    ; Command !preferences];
    [];
    [Label (s_ "Close Window")      ; Command close];
    [];
    [Label (s_ "Quit")              ; Command really_quit]
  ];
```

⟨*signature* About.f 44c⟩≡ (423d)
```
  val f : unit -> unit
```

⟨*function* About.f 44d⟩≡ (425b)
```
  let f () =
    Frx_dialog.f Widget.default_toplevel
      (Mstring.gensym "about") "About MMM"
      (Version.about (Lang.lang ()))
      (Tk.Predefined "info")
      0
      ["Thanks"] |> ignore
```

⟨*function* Mmm.navigator.open_sel 44e⟩≡ (35a)
```
  let open_sel () =
    try
      let url = Selection.get [] in
      Nav.absolutegoto caps nav url
    with _ -> ()
  in
```

⟨*function* Mmm.navigator.open_file 45a⟩≡ (35a)
```
  let open_file () =
    Fileselect.f (s_ "Open File") (function
      | [] -> ()
      | [s] ->
          let path = Msys.tilde_subst s in
          Nav.absolutegoto caps nav ("file://localhost/"^path)
      | _l -> raise (Failure "multiple selection")
     )
       "*"
       ""
       false
       false
  in
```

⟨*function* Mmm.navigator.save 45b⟩≡ (35a)
```
  let save () =
    Save.document hist.h_current.h_did None
  in
```

⟨*signature* Save.document 45c⟩≡ (335d)
```
  val document : Document.id -> string option -> unit
```

⟨*function* Mmm.navigator.print 45d⟩≡ (35a)
```
  let print () =
    Save.document hist.h_current.h_did (Some (sprintf "|%s" !Save.print_command))
  in
```

⟨*function* Mmm.navigator.close 45e⟩≡ (35a)
```
  let close () =
    if !navigators = 1
    then quit true
    else Tk.destroy top
  in
```

⟨*function* Mmm.navigator.really_quit 45f⟩≡ (35a)
```
  let really_quit () =
    quit false
  in
```

⟨*function* Mmm.quit 45g⟩≡ (460)
```
  let quit (confirm : bool) =
    if confirm then
      match
       Frx_dialog.f Widget.default_toplevel (Mstring.gensym "quit")
        (s_ "Confirm")
        (s_ "Do you really want to quit ?")
        (Predefined "question")
        0
        [s_ "Yep"; s_ "Nope"]
      with
      | 0 -> Tk.destroy Widget.default_toplevel
      | _ -> ()
    else Tk.destroy Widget.default_toplevel
```

5.6.2 Navigation menu

⟨Mmm.navigator() *Navigation menu* 46a⟩≡ (44a)
```
  (* Navigation menu *)
  let navb = Menubutton.create_named mbar "navigate" [Text (s_ "Navigate")] in
  let navm = Menu.create_named navb "menu" [] in
  Menubutton.configure navb [Menu navm];

  configure_menu_elements navm [
    [Label (s_ "Home");    Command gohome];
    [Label (s_ "Back");    Command back];
    [Label (s_ "Forward"); Command forward];
    []
  ];
```

⟨Mmm.navigator() *History menu* 46b⟩≡ (44a)
```
  (* The history menu is destroyed and rebuild each time.
   * Deleting all entries will cause a callback leak since
   * entries are associated to the menu itself
   *)
  Menu.add_cascade navm [Label (s_ "History")];

  let hmenu = ref (Menu.create_named navm "history" []) in
  update_vhistory := (fun () ->
    Tk.destroy !hmenu;
    hmenu := Menu.create_named navm "history" [];
    History.contents hist |> List.iter (fun (e : History.entry) ->
      let label =
         Url.string_of e.h_did.document_url ^
         (match e.h_fragment with None -> "" | Some f -> "#"^f) ^
         (match e.h_did.document_stamp with 0 -> "" | n ->"("^string_of_int n^")")
      in
      Menu.add_command !hmenu
         [Label label;
          Command (fun () ->
            let current = hist.h_current in
            History.set_current hist e;
            if not (Nav.historygoto caps nav e.h_did e.h_fragment true)
            then History.set_current hist current
           )
          ]
      );
      Menu.configure_cascade navm (Pattern (s_ "History")) [Menu !hmenu]
  );
```

⟨Mmm.navigator() *call* `update_vhistory` 46c⟩≡ (41a)
```
  !update_vhistory();
```

5.6.3 Document menu

⟨Mmm.navigator() *Document menu* 46d⟩≡ (44a)
```
  let docb = Menubutton.create_named mbar "document" [Text (s_ "Document")] in
  let docm = Menu.create_named docb "menu" [] in
  Menubutton.configure docb [Menu docm];

  configure_menu_elements docm [
    [Label (s_ "Abort")          ; Command abort];
    [Label (s_ "Reload")         ; Command reload];
    [Label (s_ "Update")         ; Command update_true];
    ⟨Document menu elements 169b⟩
  ];
```

5.6.4 Other menu

```
⟨Mmm.navigator() Other menu 47a⟩≡                                    (44a)
  (* Other stuff *)
  let othersb = Menubutton.create_named mbar "others" [Text (s_ "Others")] in
  let othersm = Menu.create_named othersb "menu" [] in
  Menubutton.configure othersb [Menu othersm];

  ⟨Other menu elements 229e⟩
```

5.6.5 Help menu

```
⟨Mmm.navigator() Help menu 47b⟩≡                                     (44a)
  (* Help menu *)
  let helpb = Menubutton.create_named mbar "help" [Text (s_ "Help")] in
  let helpm = Menu.create_named helpb "menu" [] in
  Menubutton.configure helpb [Menu helpm];

  ⟨Help menu elements 47c⟩
```

```
⟨Help menu elements 47c⟩≡                                       (47b)  47d▷
  Menu.add_command helpm
    [Label (s_ "Version information");
     Command (fun () ->
        Nav.absolutegoto caps nav (Version.initurl (Lang.lang ())))];
```

```
⟨Help menu elements 47d⟩+≡                                 (47b)  ◁47c  209i▷
  Menu.add_command helpm
    [Label (s_ "Home Page of MMM");
     Command (fun () ->
       navigator caps false (Lexurl.make (Version.home_mmm (Lang.lang ()))) |>ignore)];
```

```
⟨signature Version.home 47e⟩≡                                        (285e)
  val home_mmm : string -> string     (* MMM home page *)
```

```
⟨function Version.home 47f⟩≡                                         (286a)
  let home_mmm = function
    | "iso8859" -> "http://pauillac.inria.fr/mmm/"
    | _ -> assert false
```

5.7 Status log

```
⟨Mmm.navigator() navigation buttons 47g⟩+≡                 (41a)  ◁43e  202a▷
  let loggingb = Label.create_named buttons "logging"
    [TextWidth 40; TextVariable loggingv; Anchor W] in
```

```
⟨local Mmm.navigator.loggingv 47h⟩≡                                  (34c)
  let loggingv = Textvariable.create_temporary top in
```

```
⟨Nav.t logging method 47i⟩≡                                          (35b)
  nav_log : string -> unit;
```

⟨Mmm.navigator() *set nav fields* 48a⟩+≡ (34a) ◁42g 48d▷
```
  nav_log = (fun s ->
    Logs.info (fun m -> m "%s" s);
    Textvariable.set loggingv s
  );
```

⟨Nav.t *error methods* 48b⟩≡ (35b)
```
  nav_error : Error.t;   (* popping error dialogs *)
```

⟨*local object* Mmm.navigator.error 48c⟩≡ (34c)
```
  let error = new Tk_error.t top in
```

⟨Mmm.navigator() *set nav fields* 48d⟩+≡ (34a) ◁48a 203a▷
```
  nav_error = error;
```

5.8 Keyboard shortcuts

⟨Mmm.navigator() *keyboard shortcuts setting* 48e⟩≡ (34a)
```
  (* Short cuts *)

  (* All the available shortcuts functions and their short cut keys. *)
  (* If you put a new function with its short cut key here, then *)
  (* Short cut string will be displayed automatically, when these *)
  (* functions are added as menu elements. *)

  (* Sorry, we use function equality, so we cannot use lambdas in the list *)
  let update_true = fun () -> update true in

  (* The shortcuts and the default settings *)
  let all_short_cuts = [
    (* function     resource name      default key sequence *)
    About.f,        "About",           [[], KeyPressDetail "F1"];
    new_window,     "NewWindow",       [[Alt], KeyPressDetail "n"];
    open_sel,       "OpenSelection",   [[Alt], KeyPressDetail "y"];
    open_file,      "OpenFile",        [[Alt], KeyPressDetail "o"];
    save,           "Save",            [[Alt], KeyPressDetail "s"];
    print,          "Print",           [];
    !preferences,   "Preference",      [[Alt], KeyPressDetail "p"];
    close,          "Close",           [[Alt], KeyPressDetail "c"];
    really_quit,    "Quit",            [[Alt], KeyPressDetail "q"];

    gohome,         "Home",            [];
    back,           "Back",            [[Alt], KeyPressDetail "Left"];
    forward,        "Forward",         [[Alt], KeyPressDetail "Right"];
    reload,         "Reload",          [[Alt], KeyPressDetail "r"];
    abort,          "Abort",           [[], KeyPressDetail "Escape"];

    update_true,    "Update",          [[Alt], KeyPressDetail "u"];
    redisplay,      "Redisplay",       [[Control], KeyPressDetail "l"];
    add_to_hotlist, "AddToHotlist",    [[Alt], KeyPressDetail "a"];
    load_images,    "LoadImages",      [[Alt], KeyPressDetail "i"];
    view_source,    "ViewSource",      [[Alt], KeyPressDetail "e"]
  ]
  in

  (* Real shortcuts information actually used *)
  let my_short_cuts =
   all_short_cuts |> List.map (fun (f,r,d) ->
    f, Tkresource.event_sequence ("shortcut" ^ r) d
   )
  in

  (* we break after each event so that All bindings, such as menu traversal,
   * dont get invoked if we destroyed the window for some reason
   * may be required only for things like reload
   *)
  my_short_cuts |> List.iter (fun (f, eventl) ->
    if eventl <> []
    then bind top eventl (BindSetBreakable ([], fun _ -> f(); break()))
  );
```

⟨*function* Mmm.navigator.configure_menu_elements 49a⟩≡ (44a)

```
  let configure_menu_elements menu xs =
    let rec list_assoc_address k = function
      | [] -> raise Not_found
      | (k',v)::_ when k == k' -> v
      | _::xs -> list_assoc_address k xs
    in
    xs |> List.iter (fun l ->
      let opts =
       List.fold_right (fun opt st ->
         (match opt with
         | Command f ->
             Command f ::
               (try
                 [Accelerator (Tkresource.short_event_sequence
                                (list_assoc_address f my_short_cuts))]
               with Not_found -> []
               )
         | _ -> [opt])
         @ st
       ) l []
      in
      match opts with
      | [] -> Menu.add_separator menu
      | _  -> Menu.add_command menu opts
   )
  in
```

5.9 Events

XXX ???

⟨*signature* Glevents.get 49b⟩≡ (285b)

```
  val get : string -> (modifier list * xEvent) list
```

⟨*signature* Glevents.reset 49c⟩≡ (285b)

```
  val reset : unit -> unit
```

⟨*constant* Glevents.events 49d⟩≡ (285c)

```
  (* A global table for describing events
   * TODO: use virtual events because here we don't change bindings in
   * place after a preference reload
   *)
  let events = Hashtbl.create 37
```

⟨*constant* Glevents.builtin_defaults 50a⟩≡ (285c)
```
let builtin_defaults = [

  (* tachymeter bindings *)
  "tachy_about", [[], ButtonPressDetail 3];
  "tachy_gc",[[], KeyPressDetail "g"; [], KeyPressDetail "c"];
  "tachy_new", [[], ButtonPressDetail 1];
  "tachy_sel", [[], ButtonPressDetail 2];

  (* bindings on inlined images *)
  "loadimage", [[Control], ButtonPressDetail 1];
  "alt_imap", [[],ButtonPressDetail 1]; (* alt mode client side img map *)
  "stopanim",  [[], ButtonPressDetail 2];
  "restartanim", [[Shift], ButtonPressDetail 2];
  "copyimgurl", [[], ButtonPressDetail 2];
  "updateimage", [[Shift], ButtonPressDetail 2];

  (* anchor bindings *)
  "goto", [[], ButtonPressDetail 1];
  "save", [[Shift], ButtonPressDetail 1];
  "gotonew", [[], ButtonPressDetail 3];
  "hypermenu", [[Control], ButtonPressDetail 1];
]
```

⟨*constant* Glevents.get 50b⟩≡ (285c)
```
let get = Hashtbl.find events
```

⟨*function* Glevents.reset 50c⟩≡ (285c)
```
(* This is for preferences *)
let reset () =
  Hashtbl.clear events;
  (* Now: for all names defined in defaults, check a possible overriding value
     in resources *)
  List.iter (fun (name,default) ->
    Hashtbl.add events
      name (Tkresource.event_sequence (sprintf "bind<%s>" name) default))
    builtin_defaults
```

Chapter 6

Parsing

6.1 URLs

⟨*signature* Lexurl.f 51a⟩≡ (288a)
```
  val f : Lexing.lexbuf -> Url.t
```

⟨*signature* Lexurl.make 51b⟩≡ (288a)
```
  val make : string -> Url.t
     (* raise Url_Lexing(msg,pos) *)
```

⟨*function* Lexurl.make 51c⟩≡ (288b)
```
  let make s =
    f (Lexing.from_string s)
```

6.1.1 Protocol

⟨*function* Lexurl.f 51d⟩≡ (288b)
```
  (* We don't actually need all of this *)
  rule f = parse
    [ 'a'-'z' 'A'-'Z' '0'-'9' '+' '.' '-' ]+ ":"  (* absolute url *)
      { let lexeme = Lexing.lexeme lexbuf in
        let result =
          { protocol = HTTP; (* will be adjusted later *)
            user = None; password = None;
            host = None; port = None;
            path = None; search = None
           }
        in
        let protocol =
          String.uppercase_ascii (String.sub lexeme 0 (String.length lexeme - 1)) in
        (match protocol with
        ⟨Lexurl.f protocol cases 51e⟩
        | s ->
            result.protocol <- OtherProtocol s;
            result.path <- any lexbuf
        );
        result
      }
   | _ { raise (Url_Lexing ("not an URL", Lexing.lexeme_start lexbuf)) }
```

⟨Lexurl.f *protocol cases* 51e⟩≡ (51d) 210a ▷
```
  | "HTTP" | "HTTPS" ->
      slashslash lexbuf;
      let h, po = hostport lexbuf in
      let pa, se = pathsearch lexbuf in
      let proto =
        match protocol with
        | "HTTP" -> HTTP
        | "HTTPS" -> HTTPS
        | _ -> raise (Impossible "see match cases above")
      in
      result.protocol <- proto;
      result.host <- h;
      result.port <- normalize_port (proto, po);
      result.path <- pa;
      result.search <- se
```

⟨*function* Lexurl.slashslash 52a⟩≡ (288b)

```
  and slashslash =  parse
    "//" { () }
  | ""   { raise (Url_Lexing ("// expected", Lexing.lexeme_start lexbuf)) }
```

⟨*function* Lexurl.normalize_port 52b⟩≡ (288b)

```
  (* coupling: with Http.request where those port numbers are also used *)
  let normalize_port = function
    | HTTP, Some 80 -> None
    | HTTPS, Some 443 -> None
    | FTP, Some 21 -> None
    (* incomplete, but we don't care yet *)
    | _, p -> p
```

6.1.2 Host, port

⟨*function* Lexurl.hostport 52c⟩≡ (288b)

```
  (* _ is not legal in hostnames, but some people use it. *)
  and hostport = parse
  | ['A'-'Z' 'a'-'z' '0'-'9' '.' '-' '_']+ ':' ['0'-'9']+
      { let lexeme = Lexing.lexeme lexbuf in
        let pos = String.index lexeme ':' in
        let portstring =
          String.sub lexeme (succ pos) (String.length lexeme - 1 - pos) in
        Some (normalize_host (String.sub lexeme 0 pos)),
        Some (int_of_string portstring)
      }
  | ['A'-'Z' 'a'-'z' '0'-'9' '.' '-' '_']+
      { Some (normalize_host (Lexing.lexeme lexbuf)), None }
  | "" (* file:///home/... *)
      { None, None }
```

⟨*function* Lexurl.normalize_host 52d⟩≡ (288b)

```
  (* lowercase, don't use final . in FQDN *)
  let normalize_host s =
    let s = String.lowercase_ascii s in
    let l = String.length s in
    if s.[l-1] = '.'
    then String.sub s 0 (l-1)
    else s
```

6.1.3 Path, search

⟨*function* Lexurl.pathsearch 53a⟩≡ (288b)
```
  (* /<path>?<search> *)
  and pathsearch = parse
  | "/" [^ '?']* '?'
      { let lexeme = Lexing.lexeme lexbuf in
        let search = any lexbuf in
        Some (String.sub lexeme 1 (String.length lexeme - 2)), search
      }
  | "/" [^ '?']*
      { let lexeme = Lexing.lexeme lexbuf in
        Some (String.sub lexeme 1 (String.length lexeme - 1)), None
      }
  | ""
      { None, None }
```

⟨*functions* Lexurl.xxx 53b⟩≡ (288b) 54b▷
```
  and any = parse
    [^ '\n']*  { Some (Lexing.lexeme lexbuf) }    (* in fact any char *)
```

6.1.4 Normalization

⟨*signature* Lexurl.normalize 53c⟩≡ (288a)
```
  val normalize : string -> string
```

⟨*function* Lexurl.normalize 53d⟩≡ (288b)
```
  let normalize (url : string) : string =
    let urlp = make url in
    Url.string_of urlp
```

⟨*signature* Lexurl.maken 53e⟩≡ (288a)
```
  val maken : string -> Url.t
     (* raise Url_Lexing(msg,pos) *)
```

⟨*function* Lexurl.maken 53f⟩≡ (288b)
```
  (* Extra normalisation at lexing time
   *  remove ../ and /. as in RFC 1630
   *  unquote %
   *)
  let maken s =
    let url = make s in
    (match url.protocol with
    | HTTP ->
        (match url.path with
        | None -> ()
        | Some p -> url.path <- Some (Urlenc.unquote (remove_dots p))
        )
    | _ -> ()
    );
    url
```

dots

⟨*signature* Lexurl.remove_dots 53g⟩≡ (288a)
```
  val remove_dots : string -> string
```

⟨*function* Lexurl.remove_dots 54a⟩≡ (288b)

```
  let remove_dots s =
    let b = Ebuffer.create 32 in
    rev_do_list
      (Ebuffer.output_string b)
      (pathcomponents (Lexing.from_string s) []);
    Ebuffer.get b
```

⟨*functions* Lexurl.xxx 54b⟩+≡ (288b) ◁53b 210b▷

```
  and pathcomponents = parse
    [ ^ '/']* '/'
      { (fun l ->
           let newl =
             match Lexing.lexeme lexbuf with
            | "./" -> l
             | "../" -> (match l with | [] -> [] | _ :: tl -> tl)
             | p -> (p :: l)
            in
          pathcomponents lexbuf newl)
       }
  | [ ^ '/']+
      { (fun l ->
          match Lexing.lexeme lexbuf with
          | "." -> l
          | ".." -> (match l with [] -> [] | _ :: tl -> tl)
          | p -> p :: l
         )
      }
  | "" { (fun l -> l) }
```

Percents

⟨*signature* Urlenc.unquote 54c⟩≡ (287b)

```
  (*-*)
  val unquote : string -> string
```

⟨*function* Urlenc.unquote 54d⟩≡ (287c)

```
  let unquote s =
    try
      (* optim *)
      let _ = String.index s '%' in
      let l = String.length s in
      let target = Ebuffer.create l in
      let pos = ref 0 in
      (try
        while !pos < l do
         let perpos = String.index_from s !pos '%' in
         if perpos > !pos
         then Ebuffer.output target s !pos (perpos - !pos);
         pos := perpos;
         if s.[!pos] = '%' && !pos + 2 < l
         then begin
           let c = 16 * hex_to_dec s.[!pos+1] + hex_to_dec s.[!pos+2] in
           let substc = Char.chr c in
           if List.mem substc keep_quoted
           then
             for _i = 0 to 2 do
               Ebuffer.output_char target s.[!pos];
               incr pos
```

```
            done
          else begin
              Ebuffer.output_char target (Char.chr c);
              pos := !pos + 3
          end
        end else begin
         Ebuffer.output_char target s.[!pos];
         incr pos
        end
      done;
      Ebuffer.get target
    with Not_found -> (* no more substitutions *)
      Ebuffer.output target s !pos (l - !pos);
      Ebuffer.get target
    )
   with Not_found -> s
```

⟨*constant* `Urlenc.keep_quoted` 55a⟩≡ (287c)

```
  (* Unquote an url path:
     We decode all % except those corresponding to significative
     characters for parsing: /, ?, #, sp, :
   *)
  let keep_quoted =
    ['/'; '?'; '#'; ' '; '\t'; '\r'; '\n'; ':'; '%'; '&'; '='; '+']
```

6.1.5 Encoding

Decoding

⟨*signature* `Urlenc.decode` 55b⟩≡ (287b)

```
  (* URL encoding *)
  val decode : string -> string
```

⟨*function* `Urlenc.decode` 55c⟩≡ (287c)

```
  (* Decode escaped characters *)
  (* Note: beware of order of splitting wrt '&' and decoding *)
  let decode s =
    let l = String.length s in
    let target = Ebuffer.create l in
    let pos = ref 0 in
    while !pos < l do
      if s.[!pos] = '%' && !pos + 2 < l  then begin
         let c = 16 * hex_to_dec s.[!pos+1] + hex_to_dec s.[!pos+2] in
         Ebuffer.output_char target (Char.chr c);
      pos := !pos + 3
      end else if s.[!pos] = '+' then begin
        Ebuffer.output_char target ' ';
        incr pos
      end else begin
        Ebuffer.output_char target s.[!pos];
        incr pos
        end
    done;
    Ebuffer.get target
```

Encoding

⟨*signature* `Urlenc.encode` 55d⟩≡ (287b)

```
  val encode : string -> string
      (* encoding and decoding for an arbitrary string *)
```

⟨*function* Urlenc.encode 56a⟩≡ (287c)
```
  let encode s =
    let target = Ebuffer.create (String.length s) in
    for pos = 0 to String.length s - 1 do
      match s.[pos] with
        ' ' -> Ebuffer.output_char target '+'
      | '0'..'9' | 'a'..'z' | 'A'..'Z' as c -> Ebuffer.output_char target c
      | '\n' -> Ebuffer.output_string target "%0D%0A"
      | c -> Ebuffer.output_string target (hexchar c)
      done;
    Ebuffer.get target
```

⟨*function* Urlenc.hexchar 56b⟩≡ (287c)
```
  let hexchar c =
    let s = Bytes.make 3 '%'
    and i = Char.code c in
    Bytes.set s 1 (dec_to_hex (i/16));
    Bytes.set s 2 (dec_to_hex (i mod 16));
    Bytes.to_string s
```

Forms

⟨*signature* Urlenc.form_decode 56c⟩≡ (287b)
```
  val form_decode : string -> (string * string) list
      (* application/x-www-form-urlencoded encoding *)
```

⟨*constant* Urlenc.form_decode 56d⟩≡ (287c)
```
  let form_decode =
    let ampersand c = c = '&' and equals c = c = '=' in
    (function  s ->
       List.map (fun encp ->
         match split_str equals encp with
         [x;y] -> (decode x, decode y)
       | [x] -> (decode x, "")
       | _ -> invalid_arg "form_decode")
         (split_str ampersand s))
```

⟨*signature* Urlenc.form_encode 56e⟩≡ (287b)
```
  val form_encode : (string * string) list -> string
```

⟨*function* Urlenc.form_encode 56f⟩≡ (287c)
```
  let form_encode = function
   | [] -> ""
   | (e,v)::l ->
      let b = Ebuffer.create 512 in
      Ebuffer.reset b;
      Ebuffer.output_string b (encode e);
      Ebuffer.output_char b '=';
      Ebuffer.output_string b (encode v);
      l |> List.iter (fun (e,v) ->
           Ebuffer.output_char b '&';
           Ebuffer.output_string b
              (if !strict_form_standard
               then encode e
               else e
               );
           Ebuffer.output_char b '=';
           Ebuffer.output_string b (encode v)
      ) ;
      Ebuffer.get b
```

⟨*signature* Urlenc.strict_form_standard 57a⟩≡ (287b)

```
  val strict_form_standard : bool ref
      (* if true, we take RFC1866 8.2.1 case 1 strictly, and encode any
         non-alphanumeric character in the field name
         else, we encode only values, but not field names *)
```

⟨*constant* Urlenc.strict_form_standard 57b⟩≡ (287c)

```
  let strict_form_standard = ref true
```

6.2 Links

⟨*signature* Hyper.resolve 57c⟩≡ (288c)

```
  val resolve : link -> Uri.abs_uri
    (* raises Invalid_link(msg) *)
```

⟨*function* Hyper.resolve 57d⟩≡ (289a)

```
  (* Produces an URI *)
  let resolve (link : link) : Uri.abs_uri =
    (* First remove the possible fragment of the uri *)
    let newuri, frag =
      try
         let pos = String.index link.h_uri '#' in
         String.sub link.h_uri 0 pos,
         Some (String.sub link.h_uri (succ pos)
                           (String.length link.h_uri - pos - 1))
      with Not_found -> link.h_uri, None
    in
    if Uri.is_absolute newuri
    then
      try
        { uri_url = Lexurl.normalize newuri;
          uri_fragment = frag
        }
      with Url_Lexing _ ->
        raise (Invalid_link (LinkResolve (s_ "not a legal absolute uri")))

    else begin (* It is a relative uri *)
      let context =
        match link.h_context with
        | None ->
           raise (Invalid_link
                   (LinkResolve (s_ "no context and not an absolute url")))
        | Some c -> c
      in
      let contextp =
         try Lexurl.maken context
         with Url_Lexing (err,pos) ->
               raise (Invalid_link (UrlLexing (err,pos)))
      in
      { uri_url = urlconcat contextp newuri;
        uri_fragment = frag
      }
    end
```

⟨*signature* Hyper.urlconcat 57e⟩≡ (288c)

```
  val urlconcat: Url.t -> string -> string
     (* [urlconcat url relurl] resolves the relative URL [relurl] in the
         context of the URL [url]
        Doesn't handle fragments
      *)
```

⟨*function* Hyper.urlconcat 58a⟩≡ (289a)
```
(* parsed Absolute URL + URL -> Absolute URL *)
(* NO FRAGMENT HANDLING *)

let urlconcat (contextp : Url.t) (newuri : string) : string =
  let l = String.length newuri in
  if l = 0
  then string_of contextp
  else
   if l > 2 && newuri.[0] = '/' && newuri.[1] = '/'
   then
      (* this is probably a gopher relative uri *)
      sprintf "%s:%s" (string_of_protocol contextp.protocol) newuri
   else
     if newuri.[0] = '/'
     then (* start from root *)
       string_of { contextp with
                   path = Some (Urlenc.unquote
                      (String.sub newuri 1 (String.length newuri - 1)));
                   search = None }
     else
       if newuri.[0] = '?'
       then (* change only search part *)
         string_of { contextp with
                     search = Some(String.sub newuri 1(String.length newuri-1))}
       else
        let pathpart,searchpart =
         try
           let n = String.index newuri '?' in
           String.sub newuri 0 n,
           Some (String.sub newuri (n+1) (l - n - 1))
         with Not_found -> newuri, None
        in
        match contextp.path with
        | None | Some "" ->
            string_of { contextp with
                        path=Some(Urlenc.unquote(Lexurl.remove_dots pathpart));
                        search = searchpart }
        | Some old ->
            (* only the "dirname" part of the context path is important *)
            (* e.g  .../d/e/f becomes /d/e/ *)
            let path = sprintf "%s/%s" (Filename.dirname old) pathpart in
            (* we then have to remove dots *)
            let reduced = Lexurl.remove_dots path in
            string_of { contextp with
                        path = Some (Urlenc.unquote reduced);
                        search = searchpart }
```

⟨*signature* Hyper.parse_method 58b⟩≡ (288c)
```
val parse_method : string -> link_method
```

⟨*function* Hyper.parse_method 58c⟩≡ (289a)
```
let parse_method = function
  | "GET" -> GET
  | "POST" -> POST ""
  | "HEAD" -> HEAD
  | _ -> raise Not_found (* other cases should be caught by caller ! *)
```

6.3 HTML

6.3.1 Lexing

⟨*signature* Lexhtml.html 59a⟩≡ (296a)
```
  val html : Lexing.lexbuf -> t -> warnings * Html.token * Html.location
```

⟨*signature* Lexhtml.cdata 59b⟩≡ (296a)
```
  val cdata : Lexing.lexbuf -> t -> warnings * Html.token * Html.location
```

⟨*type* Html.location 59c⟩≡ (295 293d)
```
  type location = Loc of int * int
```

⟨*type* Lexhtml.warnings 59d⟩≡ (296)
```
  type warnings = (string * int) list
```

Reentrant lexers

⟨*signature type Lexhtml.t* 59e⟩≡ (296a)
```
  type t
```

⟨*signature* Lexhtml.new_data 59f⟩≡ (296a)
```
  val new_data : unit -> t
      (* instance data for a lexer; must be allocated for each instance, in
         order to get reentrant lexers
       *)
```

⟨*type* Lexhtml.t 59g⟩≡ (296b)
```
  (* Smart hack to make lexers reentrant.
   * Make each action a function taking "private" data as argument.
   * Invoke each action with additionnal argument.
   *
   * This works only because calls to actions in csllex generated code
   * are terminal.
   *)
  type t = {
    buffer : Ebuffer.t;
    mutable start : int;
    (*mutable*) pos_fix : int
  }
```

⟨*function* Lexhtml.new_data 59h⟩≡ (296b)
```
  let new_data () = {
    buffer = Ebuffer.create 512;
    start = 0;
    pos_fix = 0
  }
```

Helpers

⟨*helper functions* Lexhtml.xxx 59i⟩≡ (296b) 59j ▷
```
  let noerr = []
```

⟨*helper functions* Lexhtml.xxx 59j⟩+≡ (296b) ◁ 59i
```
  let mk_start lexbuf lexdata =
    Lexing.lexeme_start lexbuf - lexdata.pos_fix
  let mk_end lexbuf lexdata =
    Lexing.lexeme_end lexbuf - lexdata.pos_fix
  let mk_loc lexbuf lexdata =
    Loc (mk_start lexbuf lexdata, mk_end lexbuf lexdata)
```

⟨*function* Lexhtml.html 60a⟩≡ (296b)
```
  rule html = parse
  ⟨Lexhtml.html() rule cases 60b⟩
   | "\r\n"
      { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         Ebuffer.reset lexdata.buffer;
         Ebuffer.output_char lexdata.buffer '\n';
         text lexbuf lexdata )}
   | "\r"
      { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         Ebuffer.reset lexdata.buffer;
         Ebuffer.output_char lexdata.buffer '\n';
         text lexbuf lexdata )}
  | eof
      { (fun lexdata ->
          (noerr, EOF, mk_loc lexbuf lexdata)) }
  | _ { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         Ebuffer.reset lexdata.buffer;
         Ebuffer.output_char lexdata.buffer (Lexing.lexeme_char lexbuf 0);
         text lexbuf lexdata )}
```

Comments

⟨Lexhtml.html() *rule cases* 60b⟩≡ (60a) 60c ▷
```
  | "<!>"
      { (fun lexdata ->
         noerr, Comment "", mk_loc lexbuf lexdata)}
```

⟨Lexhtml.html() *rule cases* 60c⟩+≡ (60a) ◁ 60b 61d ▷
```
  (* If you think it is possible to deal with malformed comments adaptatively,
     that is switching to lenient mode only after we detected an error
     in comment syntax, then ponder the following example: <!-- -- --> *)
   | "<!--"
      { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         Ebuffer.reset lexdata.buffer;
         if !strict
         then comment lexbuf lexdata
         else lenient_end_comment lexbuf lexdata
         )
      }
```

⟨*function* Lexhtml.lenient_end_comment 60d⟩≡ (296b)
```
  (* call this ONLY if we are not in strict mode *)
  and lenient_end_comment = parse
  | "-->"
      {(fun lexdata ->
        noerr, Comment (Ebuffer.get lexdata.buffer),
        Loc(lexdata.start, mk_end lexbuf lexdata))}
  | _
      {(fun lexdata ->
        Ebuffer.output_char lexdata.buffer (Lexing.lexeme_char lexbuf 0);
        lenient_end_comment lexbuf lexdata )}
  | ""
      {(fun lexdata ->
         raise (Html_Lexing ("unterminated comment", mk_start lexbuf lexdata))
      )}
```

⟨*function* Lexhtml.comment 61a⟩≡ (296b)
```
  (* we're looking for the end of a comment : skip all characters until next   *)
  (*  -- included, and then look for next -- or > *)
  and comment = parse
    (* normal case *)
  | "--"
      { (fun lexdata ->
         next_comment lexbuf lexdata)}
  | _
      { (fun lexdata ->
         Ebuffer.output_char lexdata.buffer (Lexing.lexeme_char lexbuf 0);
         comment lexbuf lexdata )}
  | ""
      { (fun lexdata ->
         raise (Html_Lexing ("unterminated comment", mk_start lexbuf lexdata))
      )}
```

⟨*function* Lexhtml.next_comment 61b⟩≡ (296b)
```
  (* the normal next comment search *)
  and next_comment = parse
      [' ' '\t' '\r' '\n']* "--"
      { (fun lexdata -> comment lexbuf lexdata )}
    | [' ' '\t' '\r' '\n']* '>'
      { (fun lexdata ->
        [],
        Comment (Ebuffer.get lexdata.buffer),
        Loc(lexdata.start, mk_end lexbuf lexdata))}
    | ""
      { (fun lexdata ->
        raise (Html_Lexing ("invalid comment", mk_start lexbuf lexdata)))
       }
```

Tags

⟨*type* Lexhtml.tagtoken 61c⟩≡ (296b)
```
  type tagtoken =
   | Attribute of string * string
   | Closetag of int
   | Bogus of string * int   (* Bogus(s,n) == bug at pos [n] for reason [s] *)
```

⟨Lexhtml.html() *rule cases* 61d⟩+≡ (60a) ◁60c 62a▷
```
  | '<'
      { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         opentag lexbuf lexdata
       )}
```

⟨*function* Lexhtml.opentag 61e⟩≡ (296b)
```
  (* TODO 2.0:
   *   syntax for SHORTTAG YES (need to know the DTD for this !).
   *
   *)
  and opentag = parse
  | ['a'-'z' 'A'-'Z' '0'-'9' '.' '-']+
      { (fun lexdata ->
         let tagname = String.lowercase_ascii (Lexing.lexeme lexbuf) in
         let attribs = ref [] in
         let bugs = ref [] in
         let rec read_attribs () =
           match attrib lexbuf lexdata with
           | Closetag n ->
               n
           | Attribute(p1, p2) ->
               attribs := (p1, p2) :: !attribs;
               read_attribs()
           | Bogus (reason,pos) ->
               bugs := (reason,pos) :: !bugs;
               read_attribs()
         in
         let e = read_attribs() in
         (!bugs,
         OpenTag {tag_name = tagname; attributes = List.rev !attribs },
         Loc(lexdata.start, e)
         )
        )}

  (* Tolerance *)
  | ""
      { (fun lexdata ->
         Ebuffer.reset lexdata.buffer;
         Ebuffer.output_char lexdata.buffer '<';
         text lexbuf lexdata )}
```

⟨Lexhtml.html() *rule cases* 62a⟩+≡ (60a) ◁61d 65a▷

```
  | '\n'? "</"
      { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         closetag lexbuf lexdata
       )}
```

⟨*function* Lexhtml.closetag 62b⟩≡ (296b)

```
  and closetag = parse
    | ['a'-'z' 'A'-'Z' '0'-'9' '.' '-']+
      { (fun lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
      let e = skip_to_close lexbuf lexdata in
      [],
      CloseTag (String.lowercase_ascii lexeme), Loc(lexdata.start,e))}
  (* Tolerance *)
    | ""
      { (fun lexdata ->
           Ebuffer.reset lexdata.buffer;
           Ebuffer.output_string lexdata.buffer "</";
           text lexbuf lexdata)}
```

⟨*function* Lexhtml.skip_to_close 62c⟩≡ (296b)

```
  and skip_to_close = parse
     [^'>']* '>' { (fun lexdata -> mk_end lexbuf lexdata)}
    | "" { (fun lexdata ->
        raise (Html_Lexing ("unterminated tag",
                mk_start lexbuf lexdata)) )}
```

Attributes

⟨*function* Lexhtml.attrib 62d⟩≡ (296b)

```
  and attrib = parse
  | [' ' '\t' '\n' '\r']+
      { (fun lexdata -> attrib lexbuf lexdata )}
  | ['a'-'z' 'A'-'Z' '0'-'9' '.' '-']+
      { (fun lexdata ->
          let name = String.lowercase_ascii (Lexing.lexeme lexbuf) in
          try
            match tagattrib lexbuf lexdata with
            | Some s -> Attribute (name, s)
            | None -> Attribute (name, name)
          with Html_Lexing(reason,pos) ->
            if !strict
            then raise (Html_Lexing(reason,pos));
            Bogus(reason,pos)
        )}
    (* added '_' so we can parse Netscape bookmark files,
        but it should NOT be there *)
  | ['a'-'z' 'A'-'Z' '0'-'9' '.' '-' '_']+
      { (fun lexdata ->
         let name = String.lowercase_ascii (Lexing.lexeme lexbuf) in
         if !strict
         then raise (Html_Lexing ("illegal attribute name: " ^ name,
                                  mk_start lexbuf lexdata))
         else
           try
             match tagattrib lexbuf lexdata with
             | Some s -> Attribute (name, s)
             | None -> Attribute (name, name)
           with Html_Lexing(reason,pos) -> Bogus(reason,pos)
        )}
  | '>' '\n'?
      { (fun lexdata -> Closetag (mk_end lexbuf lexdata) )}
  | eof
      { (fun lexdata -> raise (Html_Lexing ("unclosed tag",
                                mk_start lexbuf lexdata)))}

  (* tolerance: we are expecting an attribute name, but can't get any *)
  (* skip the char and try again. (The char cannot be > !) *)
  | _
      { (fun lexdata ->
          if !strict
          then raise (Html_Lexing ("invalid attribute name",
                                   mk_start lexbuf lexdata));
          Bogus ("invalid attribute name", mk_start lexbuf lexdata)
      )}
```

⟨*function* Lexhtml.tagattrib 63a⟩≡ (296b)

```
  and tagattrib = parse
  | [' ' '\t' '\n' '\r']* '=' [' ' '\t' '\n' '\r']*
      { (fun lexdata -> Some (attribvalue lexbuf lexdata) )}
  | ""
      { (fun _lexdata -> None )}
```

⟨*function* Lexhtml.attribvalue 63b⟩≡ (296b)

```
  (* This should be dependent on the attribute name *)
  (* people often forget to quote, so try to do something about it *)
  (* but if a quote is not closed, you are dead *)
  and attribvalue = parse
  | ['a'-'z' 'A'-'Z' '0'-'9' '.' '-']+
      { (fun _lexdata -> Lexing.lexeme lexbuf )}
  | '"'
      { (fun lexdata ->
         Ebuffer.reset lexdata.buffer;
```

```
        inquote lexbuf lexdata )}
  | '\''
      { (fun lexdata ->
         Ebuffer.reset lexdata.buffer;
         insingle lexbuf lexdata )}
  | ""
      { (fun _lexdata ->
          raise (Html_Lexing ("illegal attribute val",
                              Lexing.lexeme_start lexbuf)) )}
```

⟨*function* Lexhtml.inquote 64a⟩≡ (296b)

```
  and inquote = parse
  | [^ '"' '&' '\027']+
      { (fun lexdata ->
         Ebuffer.output_string lexdata.buffer (Lexing.lexeme lexbuf);
         inquote lexbuf lexdata )}
  | '"'
      { (fun lexdata ->
         Html.beautify true (Ebuffer.get lexdata.buffer) )}
  | '&'
      { (fun lexdata ->
         Ebuffer.output_string lexdata.buffer (ampersand lexbuf lexdata);
         inquote lexbuf lexdata )}
  | ""
      { (fun lexdata ->
         raise (Html_Lexing ("unclosed \"", mk_start lexbuf lexdata))
       )}
```

⟨*function* Lexhtml.insingle 64b⟩≡ (296b)

```
  and insingle = parse
  | [^ '\'' '&']+
      { (fun lexdata ->
         Ebuffer.output_string lexdata.buffer (Lexing.lexeme lexbuf);
         insingle lexbuf lexdata )}
  | '\''
      { (fun lexdata ->
         Html.beautify true (Ebuffer.get lexdata.buffer) )}
  | '&'
      { (fun lexdata ->
         Ebuffer.output_string lexdata.buffer (ampersand lexbuf lexdata);
         insingle lexbuf lexdata )}
  | ""
      { (fun lexdata ->
         raise (Html_Lexing ("unclosed '", mk_start lexbuf lexdata))
      )}
```

⟨*signature* Html.beautify 64c⟩≡ (293d)

```
  val beautify: bool -> string -> string
    (* [beautify remove_leading_space s] removes sequences of SP *)
```

⟨*function* Html.beautify 64d⟩≡ (295)

```
  (*
   * Remove sequences of white
   *   turns out to be faster than global_replace in libstr
   *   could use String.blit to avoid char copying
   * NOTE: add \0 detection here (we need it for Tk)
   *)
  let beautify remove_leading (s : string) =
    let s2 = Bytes.of_string s in
    let j = ref 0 in
    let white = ref remove_leading in
    for i = 0 to String.length s - 1 do
      match s.[i] with
      | ' ' | '\t' | '\r' | '\n' | '\000' ->
          if not !white
          then begin
            Bytes.set s2 !j ' ';
            incr j;
            white := true
          end
      | c ->
          Bytes.set s2 !j c;
          white := false;
          incr j
    done;
    Bytes.sub_string s2 0 !j
```

Doctype

⟨Lexhtml.html() *rule cases* 65a⟩+≡ (60a) ◁62a 66b▷

```
  | "<!" ['D''d']['O''o']['C''c']['T''t']['Y''y']['P''p']['E''e'] [^ '>']* '>'
      { (fun lexdata ->
          noerr, Doctype (Lexing.lexeme lexbuf), mk_loc lexbuf lexdata )}
```

Text

⟨*function* Lexhtml.text 65b⟩≡ (296b)

```
  and text = parse
  | [^ '<' '&' '\r' '\027']+
      { (fun lexdata ->
        let _sTODO = Lexing.lexeme lexbuf in
        Ebuffer.output_string lexdata.buffer (Lexing.lexeme lexbuf);
        text lexbuf lexdata )}
  | [^ '<' '&' '\r' '\027']* '&'
      { (fun lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          Ebuffer.output lexdata.buffer lexeme 0 (String.length lexeme -1) ;
          Ebuffer.output_string lexdata.buffer (ampersand lexbuf lexdata);
          text lexbuf lexdata )}
  | [^ '<' '&' '\r' '\027']* '\r' '\n'
      { (fun lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          Ebuffer.output lexdata.buffer lexeme 0 (String.length lexeme - 2);
          Ebuffer.output_char lexdata.buffer '\n';
          text lexbuf lexdata )}
  | [^ '<' '&' '\r' '\027']* '\r'
      { (fun lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          Ebuffer.output lexdata.buffer lexeme 0 (String.length lexeme - 1);
          Ebuffer.output_char lexdata.buffer '\n';
          text lexbuf lexdata )}
  | ""
      { (fun lexdata ->
          noerr,
          PCData (Ebuffer.get lexdata.buffer),
          Loc(lexdata.start, mk_end lexbuf lexdata)
        )}
   (* no default case needed *)
```

Data

⟨*function* Lexhtml.cdata 66a⟩≡ (296b)
```
  and cdata = parse
  | [^ '<']* (['<']+ [^ '/']) ?
      { (fun lexdata ->
          noerr, CData(Lexing.lexeme lexbuf), mk_loc lexbuf lexdata)}

  | "</" ['a'-'z' 'A'-'Z' '0'-'9' '.' '-']+
      { (fun lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          let _eTODO = skip_to_close lexbuf lexdata in
          noerr,
          CloseTag (String.lowercase_ascii
                       (String.sub lexeme 2 (String.length lexeme - 2))),
          mk_loc lexbuf lexdata)}
  | "</"
      { (fun lexdata ->
          noerr, CData(Lexing.lexeme lexbuf), mk_loc lexbuf lexdata) }

  | eof
      { (fun lexdata ->
          noerr, EOF, mk_loc lexbuf lexdata) }
```

Entities, &xxx;

⟨Lexhtml.html() *rule cases* 66b⟩+≡ (60a) ◁65a
```
   | '&'
      { (fun lexdata ->
         lexdata.start <- mk_start lexbuf lexdata;
         Ebuffer.reset lexdata.buffer;
         Ebuffer.output_string lexdata.buffer
           (ampersand lexbuf lexdata);
         text lexbuf lexdata )}
```

⟨*function* Lexhtml.ampersand 66c⟩≡ (296b)
```
  and ampersand = parse
  | '#' ['0'-'9']+ ';'
      { (fun _lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          numeric_entity_to_utf8 (String.sub lexeme 1 (String.length lexeme - 2))
        )}
  | ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9']* ';'
      { (fun _lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          let entity = String.sub lexeme 0 (String.length lexeme - 1) in
          try
            get_entity entity
          with (* 4.2.1 undeclared markup error handling *) Not_found ->
            ("&" ^ lexeme)
        )}
    (* terminating ; is not required if next character could not be
       part of the lexeme *)
  | '#' ['0'-'9']+
      { (fun _lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          numeric_entity_to_utf8 (String.sub lexeme 1 (String.length lexeme - 1))
        )}
   | ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9']*
      { (fun _lexdata ->
          let lexeme = Lexing.lexeme lexbuf in
          try
            get_entity lexeme
          with (* 4.2.1 undeclared markup error handling *) Not_found ->
            ("&"^lexeme)
        )}
    (* Tolerance ... *)
  | ""
      { (fun _lexdata -> "&" )}
```

⟨*signature* Html.get_entity 67a⟩≡ (293d)

```
  val get_entity : string -> string
    (* [get_entity "amp"] returns "&" *)
```

⟨*constant* Html.get_entity 67b⟩≡ (295)

```
  let get_entity = Hashtbl.find ampersand_table
```

⟨*constant* Html.ampersand_table 67c⟩≡ (295)

```
  (*
   * HTML named character entities, values encoded as UTF-8
   *  cf Appendix B - Proposed Entities
   *)

  let ampersand_table =
    (Hashtbl.create 101: (string , string) Hashtbl.t)
```

⟨Main.main() *html entities initialisation* 67d⟩≡ (30c)

```
  (* Initialization of HTML entities *)
  Html.init (Lang.lang());
```

⟨*signature* Html.init 67e⟩≡ (293d)

```
  val init : string -> unit
```

⟨*function* Html.init 67f⟩≡ (295)

```
  let init _lang =
    named_entities |> List.iter (fun (str, c) ->
      Hashtbl.add ampersand_table str c)
```

⟨*constant* Html.latin1_normal 67g⟩≡ (295)

```
  let named_entities = [
    "amp",  "&";
    "gt",  ">";
    "lt" ,  "<";
    "quot",  "\"";

    ⟨latin1_normal elements 227b⟩
  ]
```

Strict mode

⟨*signature* Lexhtml.strict 67h⟩≡ (296a)

```
  val strict : bool ref
      (* if true, use strict parsing; else, activate leniency on some
         lexing decisisons such as: comments, attribute names and values
       *)
```

⟨*global* Lexhtml.strict 67i⟩≡ (296b)

```
  let strict = ref false
```

6.3.2 Parsing part 1, the DTD

⟨*module Dtd.elements* 68a⟩≡ (293a)
```
module Elements = Set.Make(struct type t = string let compare = compare end)
```

⟨*type* Dtd.t 68b⟩≡ (293a 292)
```
type t = {
  dtd_name : string;
  contents : (string, Elements.t) Hashtbl.t;
    (* for each element, give the set of included elements *)

  mutable open_omitted : Elements.t;
    (* set of elements for which opening tag may be omitted *)
  mutable close_omitted : Elements.t
    (* set of elements for which closing tag may be omitted *)
 }
```

⟨*signature* Dtd.dtd20 68c⟩≡ (292)
```
val dtd20 : t
```

⟨*signature* Dtd.dtd32 68d⟩≡ (292)
```
val dtd32 : t
```

⟨*signature* Dtd.name 68e⟩≡ (292)
```
val name : t -> string
```

⟨*function* Dtd.name 68f⟩≡ (293a)
```
let name t =
  t.dtd_name
```

⟨*function* Dtd.sol 68g⟩≡ (293a)
```
(* Utils *)
let sol l =
  List.fold_right Elements.add l Elements.empty
```

⟨*function* Dtd.sos 68h⟩≡ (293a)
```
let sos l =
  List.fold_right Elements.union l Elements.empty
```

⟨*constant* Dtd.dtd20 68i⟩≡ (293a)
```
(* #PCDATA and #CDATA are considered as elements, but they will never
   be pushed on the stack during evaluation. Moreover, since they are
   not in open_omitted/close_omitted, minimization algorithm will not
   attempt to choose them
 *)

let dtd20 =
  let dtd = {
    dtd_name = "HTML 2.0";

    contents = Hashtbl.create 53;
    open_omitted = Elements.empty;
    close_omitted = Elements.empty
  } in

  let omit_open el =
    dtd.open_omitted <- Elements.add el dtd.open_omitted in
  let omit_close el =
    dtd.close_omitted <- Elements.add el dtd.close_omitted in
  let add_elem =
    Hashtbl.add dtd.contents in
```

```
(* Some entities *)
(* <!ENTITY % heading "H1|H2|H3|H4|H5|H6"> *)
let heading_E = sol ["h1"; "h2"; "h3"; "h4"; "h5"; "h6"]
(* <!ENTITY % list " UL | OL | DIR | MENU " > *)
and list_E = sol ["ul"; "ol"; "dir"; "menu"] in
(* <!ENTITY % font " TT | B | I "> *)
let font_E = sol ["tt"; "b"; "i"]
(* <!ENTITY % phrase "EM | STRONG | CODE | SAMP | KBD | VAR | CITE "> *)
and phrase_E = sol ["em"; "strong"; "code"; "samp"; "kbd"; "var"; "cite"] in
(* <!ENTITY % text "#PCDATA | A | IMG | BR | %phrase | %font"> *)
(* EMBED added *)
let text_E =
  sos [sol ["#pcdata"; "a"; "img"; "br"; "embed"]; font_E; phrase_E] in

(* <!ELEMENT (%font;|%phrase) - - (%text)*> *)
Elements.iter (fun e -> add_elem e text_E) font_E;
Elements.iter (fun e -> add_elem e text_E) phrase_E;

(* <!ENTITY % pre.content "#PCDATA | A | HR | BR | %font | %phrase"> *)
let pre_content_E =
     sos [sol ["#pcdata"; "a"; "hr"; "br"]; font_E; phrase_E] in

(* <!ELEMENT BR    - O EMPTY> *)
add_elem "br" Elements.empty;
omit_close "br";

(* <!ENTITY % A.content   "(%heading|%text)*"> *)
let a_content_E = sos [heading_E; text_E] in

(* <!ELEMENT A     - - %A.content -(A)> *)
add_elem "a" (Elements.remove "a" a_content_E);

(* <!ELEMENT IMG    - O EMPTY> *)
add_elem "img" Elements.empty;
omit_close "img";

(* <!ELEMENT P     - O (%text)*> *)
add_elem "p" text_E;
omit_close "p";

(* <!ELEMENT HR    - O EMPTY> *)
add_elem "hr" Elements.empty;
omit_close "hr";

(* <!ELEMENT ( %heading )  - -  (%text;)*> *)
Elements.iter (fun e -> add_elem e text_E) heading_E;

(* <!ENTITY % block.forms "BLOCKQUOTE | FORM | ISINDEX"> *)
let block_forms_E = sol ["blockquote"; "form"; "isindex"] in

(* <!ENTITY % preformatted "PRE"> *)
let preformatted_E = sol ["pre"] in

(* <!ENTITY % block "P | %list | DL
    | %preformatted
    | %block.forms"> *)
let block_E = sos [sol ["p"; "dl"]; list_E; preformatted_E; block_forms_E] in

(* <!ENTITY % flow "(%text|%block)*"> *)
```

```
let flow_E = sos [text_E; block_E] in

(* <!ELEMENT PRE - - (%pre.content)*> *)
add_elem "pre" pre_content_E;


(* Deprecated but used <!ELEMENT (XMP|LISTING) - -  %literal> *)
List.iter (fun e -> add_elem e (sol ["#cdata"])) ["xmp"; "listing"];

(* <!ELEMENT DL    - -  (DT | DD)+> *)
add_elem "dl" (sol ["dt"; "dd"]);

(* <!ELEMENT DT    - O (%text)*> *)
add_elem "dt" text_E;
omit_close "dt";

(* <!ELEMENT DD    - O %flow> *)
add_elem "dd" flow_E;
omit_close "dd";

(* <!ELEMENT (OL|UL) - -  (LI)+> *)
List.iter (fun e -> add_elem e (sol ["li"])) ["ol"; "ul"];

(* <!ELEMENT (DIR|MENU) - -  (LI)+ -(%block)> *)
(* isn't that stupid ? *)
List.iter (fun e -> add_elem e (sol ["li"])) ["dir"; "menu"];

(* <!ELEMENT LI    - O %flow> *)
add_elem "li" flow_E;
omit_close "li";

(* <!ENTITY % body.content "(%heading | %text | %block |
               HR | ADDRESS)*"> *)
let body_content_E =
   sos [heading_E; text_E; block_E; sol ["hr"; "address"]] in

(* <!ELEMENT BODY O O  %body.content> *)
add_elem "body" body_content_E;
omit_open "body";
omit_close "body";

(* <!ELEMENT BLOCKQUOTE - - %body.content> *)
add_elem "blockquote" body_content_E;

(* <!ELEMENT ADDRESS - - (%text|P)*> *)
add_elem "address" (Elements.add "p" text_E);

(* <!ELEMENT FORM - - %body.content -(FORM) +(INPUT|SELECT|TEXTAREA)> *)
add_elem "form"
  (sos [Elements.remove "form" body_content_E;
    sol ["input";"select";"textarea"]]);

(* <!ELEMENT INPUT - O EMPTY> *)
add_elem "input" Elements.empty;
omit_close "input";

(* <!ELEMENT SELECT - - (OPTION+) -(INPUT|SELECT|TEXTAREA)> *)
add_elem "select" (sol ["option"]);

(* <!ELEMENT OPTION - O (#PCDATA)*> *)
```

```
add_elem "option" (sol ["#pcdata"]);
omit_close "option";

(* <!ELEMENT TEXTAREA - - (#PCDATA)* -(INPUT|SELECT|TEXTAREA)> *)
add_elem "textarea" (sol ["#pcdata"]);

(* <!ENTITY % head.extra "NEXTID? & META* & LINK*">

   <!ENTITY % head.content "TITLE & ISINDEX? & BASE? &
           (%head.extra)"> *)

let head_extra_E = sol ["nextid"; "meta"; "link"] in
let head_content_E =
  sos [sol ["title"; "isindex"; "base"]; head_extra_E] in

(* <!ELEMENT HEAD O O  (%head.content)> *)
add_elem "head" head_content_E;
omit_open "head";
omit_close "head";

(* <!ELEMENT TITLE - -  (#PCDATA)*> *)
add_elem "title" (sol ["#pcdata"]);

(* <!ELEMENT LINK - O EMPTY> *)
add_elem "link" Elements.empty;
omit_close "link";

(* <!ELEMENT ISINDEX - O EMPTY> *)
add_elem "isindex" Elements.empty;
omit_close "isindex";

(* <!ELEMENT BASE - O EMPTY> *)
add_elem "base" Elements.empty;
omit_close "base";

(* <!ELEMENT NEXTID - O EMPTY> *)
add_elem "nextid" Elements.empty;
omit_close "nextid";

(* <!ELEMENT META - O EMPTY> *)
add_elem "meta" Elements.empty;
omit_close "meta";

(* <!ENTITY % html.content "HEAD, BODY"> *)
let html_content_E = sol ["head"; "body"] in

(* <!ELEMENT HTML O O  (%html.content)> *)
add_elem "html" html_content_E;
omit_open "html";
omit_close "html";

(* fake element PCDATA for minimisation rules *)
add_elem "#pcdata" Elements.empty;

(* EMBED is an extension *)
add_elem "embed" Elements.empty;
omit_close "embed";

dtd
```

⟨*constant* Dtd.dtd32 72⟩≡ (293a)
```
  let dtd32 =
    let dtd = {
      dtd_name = "HTML 3.2";
      contents = Hashtbl.create 53;
      open_omitted = Elements.empty;
      close_omitted = Elements.empty
     } in
    let omit_open el =
      dtd.open_omitted <- Elements.add el dtd.open_omitted in
    let omit_close el =
      dtd.close_omitted <- Elements.add el dtd.close_omitted in
    let add_elem =
      Hashtbl.add dtd.contents in

    let head_misc_E = sol ["script"; "style"; "meta"; "link"]
    and heading_E = sol ["h1"; "h2"; "h3"; "h4"; "h5"; "h6"]
    and list_E = sol ["ul"; "ol"; "dir"; "menu"]
    and preformatted_E = sol ["pre"; "xmp"; "listing"]
    and font_E =
       sol ["tt"; "i"; "b"; "u"; "strike"; "big"; "small"; "sub"; "sup"]
    and phrase_E =
       sol ["em"; "strong"; "dfn"; "code"; "samp"; "kbd"; "var"; "cite"]
    and special_E =
       sol ["a"; "img"; "applet"; "font"; "basefont"; "br"; "script"; "map"]
    and form_E =
       sol ["input"; "select"; "textarea"]
    in
    (* EMBED is not in the original DTD ! *)
    let text_E =
       sos [sol ["#pcdata"; "embed"]; font_E; phrase_E; special_E; form_E]
    in
    Elements.iter (fun e -> add_elem e text_E) font_E;
    Elements.iter (fun e -> add_elem e text_E) phrase_E;
    add_elem "font" text_E;
    add_elem "basefont" Elements.empty;
    omit_close "basefont";
    add_elem "br" Elements.empty;
    omit_close "br";

    let block_E =
      sos [sol ["p"; "dl"; "div"; "center"; "blockquote"; "form"; "isindex";
                "hr"; "table"];
           list_E; preformatted_E]
    in
    let flow_E = sos [text_E; block_E] in
    let body_content_E = sos [sol ["address"]; heading_E; text_E; block_E] in
    add_elem "body" body_content_E;
    omit_open "body";
    omit_close "body";

    let address_content_E = sos [sol ["p"]; text_E] in
    add_elem "address" address_content_E;

    add_elem "div" body_content_E;
    add_elem "center" body_content_E;

    add_elem "a" (Elements.remove "a" text_E);

    add_elem "map" (sol ["area"]);
```

```
add_elem "area" Elements.empty;
omit_close "area";

add_elem "link" Elements.empty;
omit_close "link";

add_elem "img" Elements.empty;
omit_close "img";

add_elem "applet" (Elements.add "param" text_E);
add_elem "param" Elements.empty;
omit_close "param";

add_elem "hr" Elements.empty;
omit_close "hr";

add_elem "p" text_E;
omit_close "p";

Elements.iter (fun e -> add_elem e text_E) heading_E;

let pre_exclusion_E = sol ["img"; "big"; "small"; "sub"; "sup"; "font"]
in
add_elem "pre" (Elements.diff text_E pre_exclusion_E);

List.iter (fun e -> add_elem e (sol ["#cdata"])) ["xmp"; "listing"];

add_elem "blockquote" body_content_E;

add_elem "dl" (sol ["dt"; "dd"]);
add_elem "dt" text_E; omit_close "dt";
add_elem "dd" flow_E; omit_close "dd";

List.iter (fun e -> add_elem e (sol ["li"])) ["ol"; "ul"];
List.iter (fun e -> add_elem e (sol ["li"])) ["dir"; "menu"];

add_elem "li" flow_E;
omit_close "li";


add_elem "form" (Elements.remove "form" body_content_E);
add_elem "input" Elements.empty;
omit_close "input";
add_elem "select" (sol ["option"]);
add_elem "option" (sol ["#pcdata"]);
omit_close "option";
add_elem "textarea" (sol ["#pcdata"]);


add_elem "table" (sol ["caption"; "tr"]);
add_elem "tr" (sol ["th"; "td"]);
omit_close "tr";
List.iter (fun e -> add_elem e body_content_E; omit_close e) ["th"; "td"];
add_elem "caption" text_E;

let head_content_E = sol ["title"; "isindex"; "base"] in

add_elem "head" (Elements.union head_content_E head_misc_E);
omit_close "head";
omit_open "head";
```

```
    add_elem "title" (sol ["#pcdata"]);
    add_elem "isindex" Elements.empty;
    omit_close "isindex";
    add_elem "base" Elements.empty;
    omit_close "base";
    add_elem "meta" Elements.empty;
    omit_close "meta";

    add_elem "script" (sol ["#cdata"]);
    add_elem "style" (sol ["#cdata"]);

    let html_content_E = sol ["head"; "body"] in

    add_elem "html" html_content_E;
    omit_open "html";
    omit_close "html";

    (* fake element PCDATA for minimisation rules *)
    add_elem "#pcdata" Elements.empty;

    (* embed is an extension *)
    add_elem "embed" Elements.empty;
    omit_close "embed";

    dtd
```

⟨*constant* Dtd.table 74a⟩≡ (293a)

```
  let table = Hashtbl.create 11
```

⟨*signature* Dtd.add 74b⟩≡ (292)

```
  val add : t -> unit
```

⟨*signature* Dtd.get 74c⟩≡ (292)

```
  (* A table of DTDs for preferences *)
  val get : string -> t
```

⟨*signature* Dtd.names 74d⟩≡ (292)

```
  val names : unit -> string list
```

⟨*signature* Dtd.current 74e⟩≡ (292)

```
  val current : t ref
```

⟨*constant* Dtd.current 74f⟩≡ (293a)

```
  let current = ref dtd32
```

⟨*function* Dtd.add 74g⟩≡ (293a)

```
  let add t =
    Hashtbl.add table t.dtd_name t
```

⟨*constant* Dtd.get 74h⟩≡ (293a)

```
  let get =
    Hashtbl.find table
```

⟨*function* Dtd.names 74i⟩≡ (293a)

```
  let names () =
    let names = ref [] in
     Hashtbl.iter (fun name _ -> names := name :: !names) table;
     !names
```

⟨*toplevel* Dtd._1 75a⟩≡ (293a)
```
  let _ = add dtd20; add dtd32
```

6.3.3 Parsing part 2, the SGML evaluator

automat()

⟨*signature* Html_eval.automat 75b⟩≡ (297a)
```
  val automat :
    Dtd.t -> Lexing.lexbuf ->
    (Html.location -> Html.token -> unit) -> (* action callback *)
    (Html.location -> string -> unit) -> (* error callback *)
    unit
```

⟨*function* Html_eval.automat 75c⟩≡ (297b)
```
  let automat dtd lexbuf action error =
    try
      let lexer = sgml_lexer dtd in
      while true do
        try
          let warnings, correct, tokens, loc = lexer lexbuf in
          warnings |> List.iter (fun (reason, pos) ->
            error (Loc(pos,succ pos)) reason
          );
          (match correct with
          | Legal -> ()
          | Illegal reason -> error loc reason
          );
          tokens |> List.iter (function token ->
            (try
              (* calling the callback *)
              action loc token
            with Invalid_Html s -> error loc s
            );
            if token = EOF
            then failwith "quit_html_eval"
          )
        with Html_Lexing (s,n) -> error (Loc(n,n+1)) s
      done
    with Failure "quit_html_eval" -> ()
```

sgml_lexer()

⟨*type* Html_eval.minimization 75d⟩≡ (297)
```
  (* Wrapped up lexer to insert open/close tags in the stream of "normal"
     tokens, according to some DTD, in order to always get fully parenthesized
     streams *)

  type minimization =
    Legal | Illegal of string
```

⟨*signature* Html_eval.sgml_lexer 75e⟩≡ (297a)
```
  val sgml_lexer :
    Dtd.t -> Lexing.lexbuf ->
    (Lexhtml.warnings * minimization * Html.token list * Html.location)
```

⟨*function* Html_eval.sgml_lexer 76⟩≡ (297b)

```
  let sgml_lexer dtd =
    let current_lex = ref Lexhtml.html in
    let stack = ref [] in
    let lexdata = Lexhtml.new_data () in

    (* currently allowed elements *)
    let allowed () =
      match !stack with
      | [] -> initial
      | (_elem, cts)::_ -> cts
    in
    (* whatever the situation (but close), if the previous element is empty
       with an omittable close, close it *)
    let close_empty () =
      match !stack with
      | [] -> []
      | (elem, ctx)::l ->
          if Elements.is_empty ctx && Elements.mem elem dtd.close_omitted
          then (stack := l; [CloseTag elem])
          else []
    in
    (fun lexbuf ->
      let warnings, token, loc = !current_lex lexbuf lexdata in
      if !debug
      then begin prerr_string "got "; Html.print token; prerr_newline() end;
      let status, tokens =
        match token with
        | OpenTag t ->
            begin try (* first check that we know this element *)
              let contents = Hashtbl.find dtd.contents t.tag_name in
              let extraclose = close_empty() in
              (* check changing of lexers; this works only if error recovery
                 rules imply that the tag will *always* be open
               *)
              if is_cdata contents
              then current_lex := Lexhtml.cdata
              else current_lex := Lexhtml.html;

              (* is it allowed in here ? *)
              if Elements.mem t.tag_name (allowed()) then begin
                (* push on the stack *)
                stack := (t.tag_name, contents) :: !stack;
                Legal, extraclose @ [token]
              end else begin (* minimisation or error *)
                let flag, (res, l) = ominimize dtd t !stack in
                stack := l;
                flag, extraclose @ res
              end
             with Not_found ->
              (* Not in the DTD ! We return it, but don't change our state
                 or stack. An applet extension to the HTML display machine
                 can attempt to do something with it *)
              Illegal (sprintf "Element %s not in DTD" t.tag_name),
              [token]
           end

        | CloseTag t ->
            begin try (* do we know this element *)
              let _ = Hashtbl.find dtd.contents t in
              match !stack with
              | [] ->
                Illegal(sprintf "Unmatched closing </%s>" t), []
              | (elem, _cts)::l when elem = t -> (* matching close *)
                   stack := l; (* pop the stack *)
                   (* the lexer has to be "normal" again, because CDATA
                      can't be nested anyway *)
                   current_lex := Lexhtml.html;
                   Legal, [token]
              | (_elem, cts)::_l -> (* unmatched close ! *)
                  (* if we were in cdata, change the token to cdata *)
                   if is_cdata cts
                   then Legal, [CData (sprintf "</%s>" t)]
                   else begin
                     current_lex := Lexhtml.html;
                     let flag, (res, l) = cminimize dtd t !stack in
                     stack := l;
                     flag, res
                   end
          with Not_found ->
            Illegal (sprintf "Element %s not in DTD" t),
            [token]
          end
       | PCData s ->
           let extraclose = close_empty() in
           (* is it allowed in here ? *)
           if Elements.mem "#pcdata" (allowed())
           then  Legal, extraclose @ [token]
            (* ignore PCData made of spaces if not relevant to the context *)
           else
             if issp s
             then Legal, extraclose
             else begin
                (* bad hack. make believe that we try to open the #pcdata element *)
               let flag, (res, l) =
               ominimize dtd {tag_name = "#pcdata"; attributes = []} !stack in
               stack := l;
               flag,  extraclose @ res @ [token]
             end

      (* CData never happens with an empty stack *)
      | CData _s ->
          let extraclose = close_empty() in
          if Elements.mem "#cdata" (allowed())
          then Legal, extraclose @ [token]
          else
             Illegal(sprintf "Unexpected CDATA in %a" dump_stack !stack),
             extraclose @ [token]

      (* See if the stack is empty *)
      | EOF ->
         begin
          match !stack with
          | [] -> Legal, [EOF]
          | l ->
             (* we must be able to close all remaining tags *)
             let rec closethem tokens = function
             | [] -> None, EOF :: tokens
             | (last,_) :: l ->
                 if Elements.mem last dtd.close_omitted
```

```
              then closethem (CloseTag last::tokens) l
              else
                let status, tokens =
                  closethem (CloseTag last::tokens) l in
                let err = sprintf "</%s>" last in
                let newstatus =
                  match status with
                  | Some s -> Some (err^s)
                  | None -> Some err
                in
                newstatus, tokens
        in
        match closethem [] l with
        | None, tokens -> Legal, List.rev tokens
        | Some s, tokens -> Illegal ("Missing "^s), List.rev tokens
      end

    | _ ->  Legal, [token] (* ignore all other cases *)

  in
  warnings, status, tokens, loc)
```

⟨*function* Html_eval.dump_stack 78a⟩≡ (297b)

```
 let dump_stack () = function
     (x,_)::(y,_)::(z,_)::_ -> sprintf "..<%s><%s><%s>" z y x
   | [x,_;y,_] -> sprintf "<%s><%s>" y x
   | [x,_] -> sprintf "<%s>" x
   | [] -> "empty stack"
```

⟨*constant* Html_eval.initial 78b⟩≡ (297b)

```
 (* initial element of the DTD *)
 let initial = Elements.add "html" Elements.empty
```

⟨*function* Html_eval.is_cdata 78c⟩≡ (297b)

```
 let is_cdata cts =
   Elements.cardinal cts = 1 && Elements.mem "#cdata" cts
```

Minimizations

⟨*function* Html_eval.ominimize 78d⟩≡ (297b)

```
 (* open minimize
    [ominimize dtd open_tag current_stack]
    returns a list of inferred open/close tags and the new stack
  *)
 let ominimize dtd t stack =
   let elem = t.tag_name in

   (* Is elem allowed for the given stack ? *)
   let goodpos = function
       [] -> Elements.mem elem initial
     | (_, cts)::_l -> Elements.mem elem cts

   (* Return with inferred and stack.
      The stack has been reduced during the inference, so it is enough
      to push the opened element *)
   (* Special hack when t is fake #pcdata... *)
   and return inferred stack =
     if elem = "#pcdata" then
       List.rev inferred, stack
  else
    List.rev ((OpenTag t) :: inferred),
    (elem, Hashtbl.find dtd.contents elem) :: stack

in
(* [attempt_close mods_so_far current_stack] *)
let rec attempt_close accu = function
   [] -> (* reached all the possible closing, attempt to open again *)
      attempt_open accu []
 | ((last, _)::l) as stack ->
     if Elements.mem last dtd.close_omitted then
        (* we can attempt to close the previous element *)
    if goodpos l then
      (* good position, we're done *)
      return ((CloseTag last) :: accu) l
        else (* attempt to open in this new position *)
      try
            attempt_open ((CloseTag last) :: accu) l
          with
        CantMinimize -> (* try once more to close *)
           attempt_close ((CloseTag last)::accu) l
     else begin (* since we can't close, try to open *)
    attempt_open accu stack
     end

 (* [attempt_open mods_so_far currentstack] *)
 and attempt_open accu = function
   [] ->
     (* open HTML, and retry from there *)
     (* should actually iterate on all elements in initial *)
     let newstack = ["html", Hashtbl.find dtd.contents "html"]
     and newaccu = (OpenTag {tag_name = "html"; attributes = []}) :: accu
     in
    if goodpos newstack then return newaccu newstack
        else attempt_open newaccu newstack

 | ((_, cts)::_l ) as stack ->
     (* check if, in contents, there is an element with implicit omission
        that would help *)
     let possible = Elements.inter cts dtd.open_omitted in
      match Elements.cardinal possible with
    0 -> (* argh *) raise CantMinimize
      | 1 ->
    (* open this element and try from there *)
    let newelem = Elements.choose possible in
    let newaccu = (OpenTag {tag_name = newelem; attributes = []})::accu
        and newstack = (newelem, Hashtbl.find dtd.contents newelem)::stack
        in
      if goodpos newstack
      then return newaccu newstack
      else attempt_open newaccu newstack (* maybe more ? *)
      | _n -> (* since we have the choice, examine all possibilities *)
     let elems = Elements.elements possible in
     let rec backtrack = function
          [] -> raise CantMinimize
      | x::l ->
      try
        let newaccu = (OpenTag {tag_name = x; attributes = []})::accu
        and newstack = (x, Hashtbl.find dtd.contents x)::stack
                in
          if goodpos newstack then return newaccu newstack
          else attempt_open newaccu newstack
      with
       CantMinimize -> backtrack l
         in
     backtrack elems
 in
 (* now do some error recovery *)
 try Legal, attempt_close [] stack
 with
   CantMinimize ->
     (* what the hell, dammit, open it anyway, who cares, duh *)
     let _currentTODO = match stack with (x,_)::_l -> x | [] -> "" in
     Illegal (sprintf "illegal <%s> in %a, keep it though"
              t.tag_name dump_stack stack),
     return [] stack
```

⟨*function* Html_eval.cminimize 80⟩≡ (297b)

```
  (* close minimize
     [cminimize dtd elem current_stack]
    returns a list of inferred open/close tags and the new stack
   *)
  let cminimize dtd tagname stack =
    (* Is elem allowed for the given stack ? *)
    let goodpos = function
        [] -> false
      | (elem, _cts)::_l -> tagname = elem

    and return inferred stack =
       List.rev ((CloseTag tagname) :: inferred), stack

    in
    (* [attempt_close mods_so_far current_stack] *)
    let rec attempt_close accu = function
       [] -> raise CantMinimize
     | ((last, _)::l) as _stackTODO ->
         if Elements.mem last dtd.close_omitted then
            (* we can attempt to close the previous element *)
        if goodpos l then
          (* good position, we're done *)
          return (CloseTag last :: accu) (List.tl l)
            else (* close a bit more ? *)
          attempt_close ((CloseTag last)::accu) l
         else
       (* there's no reason we should have to open a new element in order
          to close the current one, is it ? *)
            raise CantMinimize
    in
    (* error recovery strategy *)
    let rec attempt_matching accu = function
       [] -> raise Not_found (* didn't find a matching open at all ! *)
     | (curelem,_):: l when curelem = tagname ->
       (* so, consider we match this open, and close them all *)
       return accu l
     | (curelem,_):: l  -> (* otherwise, find something up there *)
       attempt_matching (CloseTag curelem :: accu) l
     in
    (* now do some error recovery *)
    try Legal, attempt_close [] stack
    with
```

```
      CantMinimize ->
        try
      Illegal (sprintf "unmatched </%s> in %a, close closest match"
                     tagname dump_stack stack),
          attempt_matching [] stack
        with
      Not_found ->
        Illegal (sprintf "unmatched </%s> in %a, skipped"
                     tagname dump_stack stack),
            ([], stack) (* just skip the damn thing *)
```

⟨*exception* `Html_eval.CantMinimize` 81a⟩≡ (297b)
```
  exception CantMinimize               (* bogus HTML *)
```

Filters

⟨*signature* `Html_eval.add_html_filter` 81b⟩≡ (297a)
```
  (* test suit *)
  val add_html_filter : ((Html.token -> unit) -> Html.token -> unit) -> unit
  (* [add_html_filter filter] adds an HTML filter between the lexing and
    displaying of HTML. So, the filters do not affect the source (and
    the source display), change the content of HTML silently, and affect
    the display. The filter function [filter pfilter] receives a HTML token
    for each time, and do some job, and send a token to the parent filter
    pfilter if possible. The filters will receive a correct HTML token
    stream (all the tags are placed and closed correctly due to the DTD),
    and they must send the correct stream to the parent filter also.
  *)
```

⟨*function* `Html_eval.add_html_filter` 81c⟩≡ (297b)
```
  let add_html_filter f =
    filters := f :: !filters
```

⟨*constant* `Html_eval.filters` 81d⟩≡ (297b)
```
  let filters = ref []
```

⟨*function* `Html_eval.sgml_lexer` *(*`html/html_eval.ml`*)* 81e⟩≡ (297b)
```
  (* Redefine sgml_lexer with filters *)
  let sgml_lexer dtd =
    let org_lexer = sgml_lexer dtd in
    let buf = ref [] in
    let allfilter =
      List.fold_right (fun f st -> f st) !filters
        (fun tkn -> buf := !buf @ [tkn])
    in
    function lexbuf ->
      let warnings, correct, tokens, loc = org_lexer lexbuf in
      tokens |> List.iter allfilter; (* inefficient *)
      let tokens = !buf in
      buf := [];
      warnings, correct, tokens, loc
```

6.3.4 htparse

⟨*type* `Htparse.mode` 81f⟩≡ (298a)
```
  type mode =
   | Check
   | Indent of int
   | Nesting
```

⟨*constant* Htparse.mode 82a⟩≡ (298a)
```
let mode = ref Check
```

⟨*function* Htparse.main 82b⟩≡ (298a)
```
let main () =
  Html.init (Lang.lang());

  let options = [
     "-struct", Arg.Int   (function n -> mode := Indent n), "Parse Tree";
     "-nesting", Arg.Unit (function () -> mode := Nesting), "Check nesting";

     "-debug", Arg.Unit (function () -> Html_eval.debug := true), "Debug mode";
     "-strict", Arg.Set Lexhtml.strict, "Strict mode";
     "-v", Arg.Unit (function () -> verbose := true), "Verbose mode";

     "-dtd", Arg.Unit (function () -> Dtd.dump Dtd.dtd32f), "Dump DTD";
     "-depth", Arg.Int (function n -> Format.set_max_boxes n), "Max print depth"
     ] |> Arg.align in
    Arg.parse options
     (fun s ->
       match !mode with
       | Check -> html_lex s
       | Indent n -> html_indent s n
       | Nesting -> html_nest s
       )
     "Usage: htparse <opts> file1.html ... filen.html"
```

⟨*toplevel* Htparse._2 82c⟩≡ (298a)
```
let _ = Printexc.catch main ()
```

⟨*function* Htparse.html_lex 82d⟩≡ (298a)
```
let html_lex name =
  let ic = open_in name in
  let lexbuf, find_line = line_reporting ic in
  Html_eval.automat Dtd.dtd32f lexbuf
     (fun _loc token ->
        match token with
        | EOF -> close_in ic
        | t ->
            ⟨Htparse.html_lex() print token t if verbose 245f⟩
     )
     (error name find_line)
```

⟨*toplevel* Htparse._1 82e⟩≡ (298a)
```
let _ =
  Html.verbose := false (* we do our own error report *)
```

⟨*constant* Htparse.verbose 82f⟩≡ (298a)
```
let verbose = ref false
```

⟨*function* Htparse.error 82g⟩≡ (298a)
```
let error name find_line (Loc(n,n')) msg =
  let linenum, linestart = find_line n in
  printf "File \"%s\", line %d, characters %d-%d:\n%s\n"
         name linenum (n - linestart) (n' - linestart) msg
```

⟨*function* Htparse.line_reporting 83a⟩≡ (298a)
```
  (* lines: start at 1 *)
  (* pos: start at 0 as in caml *)
  let line_reporting ic =
    let lines = ref [] in
    let current_line = ref 1 in
    let current_pos = ref 0 in
    let read = input ic in
    Lexing.from_function (fun buf len ->
       let n = read buf 0 len in
         for i = 0 to n - 1 do
       match Bytes.get buf i with
         '\n' -> incr current_pos; incr current_line;
                 lines := (!current_pos, !current_line) :: !lines
           | _ -> incr current_pos
           done;
           n),
     (fun pos ->
       let rec find_line = function
         [] -> 1, 0
       | (linestart, _linenum)::l when pos < linestart -> find_line l
       | (linestart, linenum)::_l -> linenum, linestart
       in
        find_line !lines)
```

⟨*function* Htparse.html_nest 83b⟩≡ (298a)
```
  let html_nest name =
    let ic = open_in name in
    let lexbuf = Lexing.from_channel ic in
    let stack = ref [] in
    Html_eval.automat Dtd.dtd32f lexbuf
      (fun (Loc(n,n')) tok ->
         match tok with
         | EOF -> close_in ic
         | OpenTag t ->
             stack := t.tag_name :: !stack
         | CloseTag t ->
             (match !stack with
              | hd::tl when hd = t -> stack := tl
              | hd::_tl -> eprintf "Unmatched closing tag %s (expected %s) at
                                 pos %d - %d" t hd n n'
              | [] -> eprintf "Unmatched closing tag %s (Empty stack) at
                                 pos %d - %d" t n n'
              )
         | _ -> ()
       )
       (fun _ _ -> ())
```

⟨*function* Htparse.html_indent 83c⟩≡ (298a)
```
  let html_indent name level =
    let box =
      match level with
      | 0 -> Format.open_box
      | 1 -> Format.open_hvbox
      | _n -> Format.open_vbox
    in
    let ic = open_in name in
    let lexbuf  = Lexing.from_channel ic in
    box 0;
    Html_eval.automat Dtd.dtd32f lexbuf
```

```
      (fun _loc token ->
        match token with
        | EOF ->
            Format.print_newline();
            close_in ic
         | OpenTag t ->
            Format.print_cut();
            box 0;
            box 2;
            Format.print_string (sprintf "<%s>" t.tag_name)
         | CloseTag t ->
            Format.close_box();
            Format.print_cut();
            Format.print_string (sprintf "</%s>" t);
            Format.close_box()
        | PCData _ ->
            Format.print_string "*"
        | _ -> ()
        )
        (fun _ msg -> Format.print_string (sprintf "ERROR(%s)" msg))
```

6.4 HTTP

6.4.1 Request

⟨*signature* Http_headers.parse_request 84a⟩≡ (300g)

```
  val parse_request : string -> Messages.request_line
    (* Parses a Request-Line
       Request-Line = Method SP Request-URI SP HTTP-Version CRLF
       Raises [Invalid_HTTP_header "Request-Line"] *)
```

⟨*function* Http_headers.parse_request 84b⟩≡ (302)

```
  (* Request-Line = Method SP Request-URI SP HTTP-Version CRLF *)
  (* CHECK: Normally the URI should be encoded (no spaces ?) *)
  let parse_request s =
   try
    match Str.bounded_split (regexp "[ ]") s 3 with
      [m;r;v] ->
           { request_version = v;
             request_method = m;
             request_uri = r }
    | ["GET"; uri] ->
           { request_version = "HTTP/0.9";
             request_method = "GET";
             request_uri = uri }
    | [m;s] -> (* uri omitted ? *)
           { request_version = s;
             request_method = m;
             request_uri = "/" }
    | _ -> raise (Invalid_header "Request-Line")
   with
     Failure "int_of_string" -> raise (Invalid_header "Request-Line")
```

6.4.2 Status

⟨*signature* Http_headers.parse_status 84c⟩≡ (300g)

```
  val parse_status : string -> Messages.status_line
```

```
    (* Parses a Status-Line
       Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
       Raises [Invalid_HTTP_header "Status-Line"]
       or [Not_found] if the string is not a Status-Line at all *)
```

⟨*function* Http_headers.parse_status 85a⟩≡ (302)

```
  (* Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF *)
  let parse_status s =
   if String.length s > 5 && String.sub s 0 5 = "HTTP/"
   then
     try
      match Str.bounded_split (regexp "[ ]") s 3 with
        [v;c;m] ->
             { status_version = v;
               status_code = int_of_string c;
               status_message = m }
      (* it happened once with Server: Netscape-Commerce/1.1 *)
      (* where the Status Line was: HTTP/1.0 302 *)
      | [v;c] ->
             { status_version = v;
               status_code = int_of_string c;
               status_message = "empty" }
      | _ -> raise (Invalid_header "Status-Line")
     with Failure "int_of_string" -> raise (Invalid_header "Status-Line")
   else (* 0.9, dammit *)
     raise Not_found
```

⟨*signature* Http_headers.http_status 85b⟩≡ (300g)

```
  (* Common headers *)
  val http_status : int -> Messages.status_line
    (* [http_status n] returns Status-Line for code [n] *)
```

⟨*function* Http_headers.http_status 85c⟩≡ (302)

```
  (* A typical status line *)
  let http_status code =
    {
     status_version = "HTTP/1.0";
     status_code = code;
     status_message = status_message code
    }
```

⟨*function* Http_headers.status_message 85d⟩≡ (302)

```
  let status_message code =
    try Hashtbl.find status_messages code
    with Not_found -> " "
```

⟨*constant* Http_headers.status_messages 85e⟩≡ (302)

```
  (* Messages in Status-Line *)
  let status_messages = Hashtbl.create 101
```

⟨*toplevel* Http_headers._2 85f⟩≡ (302)

```
  let _ =
    [ 200, "OK";

      201, "Created";
      202, "Accepted";
      204, "No Content";

      301, "Moved Permanently";
      302, "Moved Temporarily";
      304, "Not Modified";

      400, "Bad Request";
      401, "Unauthorized";
      403, "Forbidden";
      404, "Not Found";

      500, "Internal Server Error";
      501, "Not Implemented";
      502, "Bad Gateway";
      503, "Service Unavailable";

      (* These are proposed for HTTP1.1 *)
      407, "Proxy Authentication Required"
    ] |> List.iter (function (code, msg) -> Hashtbl.add status_messages code msg)
```

6.4.3 Headers

⟨*signature* Http_headers.get_header 86a⟩≡ (300g)

```
  val get_header : string -> Messages.header list -> string
    (* [get_header field_name hs] returns the field_value, if any, of
       the headers, or raises [Not_found].
       [field_name] is the token is lowercase (e.g. "content-type") *)
```

⟨*function* Http_headers.get_header 86b⟩≡ (302)

```
  (* [get_header field-name headers]
   *   returns, if it exists the field value of the header field-name
   * This is a bit costly though, but we keep headers as plain strings.
   * CHECK: speed up with some regexp matching.
   * HYP: field-name in lower-case
   *)
  let get_header field_name =
    let size = String.length field_name in
    let rec search = function
     | [] -> raise Not_found
     | s::l ->
        if String.length s >= size + 2 (* : SP *) &&
           String.lowercase_ascii (String.sub s 0 size) = field_name
        then String.sub s (size + 2) (String.length s - size - 2)
        else search l
    in
    search
```

⟨*signature* Http_headers.get_multi_header 86c⟩≡ (300g)

```
  val get_multi_header : string -> Messages.header list -> string list
    (* [get_multi_header field_name hs] returns the list of field_value
       of the headers.
       [field_name] is the token is lowercase (e.g. "content-type") *)
```

⟨*function* Http_headers.get_multi_header 86d⟩≡ (302)

```
  (* [get_multi_header field_name headers]
   *   get all values of the header
   *)
  let get_multi_header field_name =
    let size = String.length field_name in (* :SP *)
    let rec search = function
       [] -> []
     | s::l ->
      if   String.length s >= size + 2 (* : SP *)
```

```
        && String.lowercase_ascii (String.sub s 0 size) = field_name
      then (String.sub s (size + 2) (String.length s - size - 2)) :: search l
      else search l in
    search
```

Content type

⟨*signature* Http_headers.contenttype 87a⟩≡ (300g)

```
  (* Predefined access functions *)
  val contenttype : Messages.header list -> string
    (* Content-Type *)
```

⟨*functions* Http_headers.xxx get_header *applications* 87b⟩≡ (302) 88e ▷

```
  let contenttype =
    get_header "content-type"
```

⟨*signature* Lexheaders.media_type 87c⟩≡ (304a)

```
  val media_type :
    string -> Http_headers.media_type * Http_headers.media_parameter list
```

⟨*function* Lexheaders.media_type 87d⟩≡ (304b)

```
  let media_type s =
    let lexbuf = Lexing.from_string s in
    let mtyp = media_type lexbuf in
    let l = media_parameters lexbuf in
    mtyp, l
```

⟨*function* Lexheaders.media_type *lexer* 87e⟩≡ (304b)

```
  (* ex: token/token *)
  and media_type = parse
  | [' ' '\t']+ {
        let _ = starlws lexbuf in
        let typ = String.lowercase_ascii (token lexbuf) in
        let _ = lit_slash lexbuf in
        let subtyp = String.lowercase_ascii (token lexbuf) in
        let _ = starlws lexbuf in (* word based *)
        typ, subtyp
      }

  | "" {
        let _ = starlws lexbuf in
        let typ = String.lowercase_ascii (token lexbuf) in
        let _ = lit_slash lexbuf in
        let subtyp = String.lowercase_ascii (token lexbuf) in
        let _ = starlws lexbuf in (* word based *)
          typ, subtyp
        }
```

⟨*function* Lexheaders.token 87f⟩≡ (304b)

```
  and token = parse
     [^ '\127'-'\255'
        '\000'-'\031'
        '(' ')' '<' '>' '@' ',' ';' ':' '\\' '"' '/' '[' ']' '?' '=' ' ' '\t']+
        { Lexing.lexeme lexbuf }
    | _ { raise (Invalid_header "token expected") }
```

⟨*function* Lexheaders.starlws 87g⟩≡ (304b)

```
  (* *LWS *)
  and starlws = parse
     ("\r\n")? [' ' '\t']+ { starlws lexbuf }
    | "" { () }
```

⟨*function* Lexheaders.media_parameters 88a⟩≡ (304b)
```
  and media_parameters = parse
  | "" { [] }
  | ";" {
       let _ = starlws lexbuf in
       let attr = String.lowercase_ascii (token lexbuf) in
       let _ = lit_equal lexbuf in (* no space allowed *)
       let v = value lexbuf in
       let _ = starlws lexbuf in
       let rest = media_parameters lexbuf in
       (attr,v)::rest
   }
```

⟨*function* Lexheaders.lit_equal 88b⟩≡ (304b)
```
  and lit_equal = parse
  | '=' { () }
  |  _  { raise (Invalid_header "= expected") }
```

⟨*function* Lexheaders.value 88c⟩≡ (304b)
```
  (* value = token | quoted-string *)
  and value = parse
  | '"' [^ '"' '\000'-'\031' '\127'-'\255' ]* '"'
      { let t = Lexing.lexeme lexbuf in
          String.sub t 1 (String.length t - 2)
      }
  | [^ '\127'-'\255'
       '\000'-'\031'
       '(' ')' '<' '>' '@' ',' ';' ':' '\\' '"' '/' '[' ']' '?' '=' ' ' '\t']+
     { Lexing.lexeme lexbuf }
  | _ { raise (Invalid_header "value expected") }
```

Content length

⟨*signature* Http_headers.contentlength 88d⟩≡ (300g)
```
  val contentlength : Messages.header list -> int
    (* Content-Length *)
```

⟨*functions* Http_headers.xxx get_header *applications* 88e⟩+≡ (302) ◁87b 88g▷
```
  let contentlength l =
    let h = get_header "content-length" l in
    try int_of_string h
    with _ -> raise Not_found
```

Content encoding

⟨*signature* Http_headers.contentencoding 88f⟩≡ (300g)
```
  val contentencoding : Messages.header list -> string
    (* Content-Encoding *)
```

⟨*functions* Http_headers.xxx get_header *applications* 88g⟩+≡ (302) ◁88e
```
  let contentencoding =
    get_header "content-encoding"
```

Chapter 7

Retrieving

⟨*signature* Retrieve.f 89a⟩≡ (318d)
```
(* f is supposed to raise only Invalid_url *)
val f : < Cap.network; .. > ->
   Www.request ->  (* the request *)
   (Hyper.link -> unit) -> (* the retry function *)
   Document.continuation -> (* the handlers *)
   status
```

⟨*type* Retrieve.retrievalStatus 89b⟩≡ (318)
```
type status =
 | Started of Www.aborter
 | InUse
```

7.1 Retrieve.f()

⟨*function* Retrieve.f 89c⟩≡ (318e)
```
(*
 * Emitting a request:
 *   we must catch here all errors due to protocols and remove the
 *   cnx from the set of active cnx.
 *)
(* Nav.request -> <> -> Http.req (via Protos.get) -> Http.tcp_connect *)
and f (caps : < Cap.network; ..>)
    (request : Www.request)
    (retry : Hyper.link -> unit)
    (cont : Document.continuation) : status =
  Logs.debug (fun m -> m "Retrieve.f on %s" (Url.string_of request.www_url));
  if Www.is_active_cnx request.www_url
  then InUse
  else begin
    Www.add_active_cnx request.www_url;
    try
      let (reqf, cachef) = Protos.get request.www_url.protocol in
      Started (reqf (caps :> < Cap.network >) request
                { cont with
                  document_process = http_check caps cachef retry cont request})

    with
    | Not_found ->
        Www.rem_active_cnx request.www_url;
        raise (Www.Invalid_request (request, s_ "unknown protocol"))
    | Http.HTTP_error s ->
        Www.rem_active_cnx request.www_url;
        raise (Www.Invalid_request (request, s_ "HTTP Error \"%s\"" s))
    | File.File_error s ->
        Www.rem_active_cnx request.www_url;
        raise (Www.Invalid_request (request, s))
    end
```

7.2 Behaviour

⟨*type* Retrieve.behaviour 90a⟩≡ (318)

```
  (* We should implement the proper behaviours for all return codes
   * defined in the HTTP/1.0 protocol draft.
   * Return codes are HTTP specific, but since all protocols are more or
   * less mapped to http, we deal with them at the retrieval level.
   *)
  type behaviour =
   | Ok                  (* process the document *)
   | Stop of string             (* stop (no document) and display message *)
   | Retry of Hyper.link        (* restart with a new link *)
   | Error of string            (* same as stop, but it's an error *)
   | Restart of (Document.handle -> Document.handle)
     (* restart the same request, but apply transformation on the continuation *)
```

7.2.1 Retrieve.http_check()

⟨*function* Retrieve.http_check 90b⟩≡ (318e)

```
  (*
   * Dispatch according to status code
   *  retry: how to re-emit a request
   *  cont: what to do with the response
   *)
  (* f -> Http.req (via protos) cont -> <> (via cont.document_process) -> codex *)
  let rec http_check (caps: < Cap.network; .. > )
           cache
           (retry : Hyper.link -> unit)
           (cont : Document.continuation)
           (wwwr : Www.request)
           (dh : Document.handle) : unit =
    Logs.debug (fun m -> m "Retrieve.http_check");
    try (* the appropriate behavior *)
      (* alt: just have a single function matching on code *)
      let behav = Hashtbl.find http_process dh.document_status in
      match behav wwwr dh with
      | Ok ->
          (* do I cache ? *)
          let cacheable = wwwr.www_link.h_method = GET in
          cont.document_process
            (if cacheable then wrap_cache cache dh else dh)

      | Stop msg ->
          Document.dclose true dh;
          cont.document_finish false;
          wwwr.www_error#ok msg
      | Error msg ->
          Document.dclose true dh;
          cont.document_finish false;
          wwwr.www_error#f msg
      | Retry hlink ->
          Document.dclose true dh;
          cont.document_finish false;
          (* Retry! will do another web request *)
          retry hlink

      | Restart transform ->
          Document.dclose true dh;
          f caps wwwr retry
           { cont with
             document_process = (fun dh -> cont.document_process (transform dh))}
          |> ignore; (* we should probably do something of the result ! *)

    with Not_found ->
       (* default behavior is to call the normal continuation
        * BUT WE DON'T CACHE !
        * e.g. 404 Not found, 500, ...
        *)
        cont.document_process dh
```

⟨*function* Retrieve.wrap_cache 91a⟩≡ (318e)

```
  (* What do we cache ? : text/html and text/plain in memory *)
  let wrap_cache cache (dh : Document.handle) : Document.handle =
    Logs.debug (fun m -> m "Wrapping cache for %s(%d)"
                (Url.string_of dh.document_id.document_url)
                dh.document_id.document_stamp);
    (* infer content-type and content-encoding using filename extension of
     * document if content-type was not specified in the headers
     *)
    Retype.f dh;
    try
      match Lexheaders.media_type (Http_headers.contenttype dh.dh_headers) with
      | ("text","html"),_
      | ("text","plain"),_ ->
        begin
         try
           let doc, c = cache dh in
           Cache.add dh.document_id
                   { document_address = dh.document_id.document_url;
                     document_data = doc;
                     document_headers = dh.dh_headers };
            Cache.wrap c dh
         with Cache.DontCache -> dh
        end
      | _ -> dh
    with Not_found -> dh
```

7.2.2 HTTP status codes

⟨*constant* Retrieve.http_process 91b⟩≡ (318e)

```
  (*
   * Provision for user (re)definition of behaviours.
   *)
  let http_process :
    (int, Www.request -> Document.handle -> behaviour) Hashtbl.t =
    Hashtbl_.create ()
```

⟨*signature* Retrieve.add_http_processor 91c⟩≡ (318d)

```
  val add_http_processor :
    int -> (Www.request -> Document.handle -> behaviour) -> unit
```

⟨*constant* Retrieve.add_http_processor 92a⟩≡ (318e)
```
  let add_http_processor = Hashtbl.add http_process
```

⟨*toplevel* Retrieve._1 92b⟩≡ (318e)
```
  (* 400 : proxies do return this code when they can satisfy the request,
   *       so we keep it as default (displayed)
   *)
  let _ =
    [200, code200;
     201, code200;
     202, code200;
     204, code204;
     ⟨Retrieve code behaviour other elements 222a⟩
    ] |> List.iter (function (code, behave) ->
       Hashtbl.add http_process code behave
    )
```

⟨*function* Retrieve.code200 92c⟩≡ (318e)
```
  (* 200 OK *)
  let code200 _wwwr _dh = Ok
  (* 201 Created (same as 200) *)
  (* 202 Accepted (same as 200) *)
```

⟨*function* Retrieve.code204 92d⟩≡ (318e)
```
  (* 204 No Content: we should modify the headers of the referer ? *)
  let code204 _wwwr (dh : Document.handle) =
    Stop (s_ "Request fulfilled.\n(%s)" (Http_headers.status_msg dh.dh_headers))
```

⟨*function* Retrieve.code400 92e⟩≡ (318e)
```
  (* 400 Bad request *)
  let _code400 _wr _dh = Error (s_ "Bad Request")
```

7.3 Active connections

⟨*module Www.UrlSet* 92f⟩≡ (291a)
```
  (* Table of unresolved active connexions *)
  (* We need to keep a trace of pending connections, since there is a race
     condition when the user clicks twice rapidly on an anchor. If the second
     click occurs before the document is added to the cache, (e.g. because we
     are waiting for the headers), then the document will be retrieved twice.
     And naturally, for documents that don't enter the cache we always will
     duplicate connexions.
     Retrieve.f is a safe place to add the request to the list of pending
     connexions, because it is synchronous.
     Removing an active connexion must take place when we close the
     dh.document_fd.
  *)
  module UrlSet = Set.Make(struct type t = Url.t let compare = compare end)
```

⟨*constant* Www.active_connexions 92g⟩≡ (291a)
```
  let active_connexions = ref UrlSet.empty
```

⟨*signature* Www.is_active_cnx 92h⟩≡ (290b)
```
  val is_active_cnx : Url.t -> bool
```

⟨*signature* Www.add_active_cnx 92i⟩≡ (290b)
```
  val add_active_cnx : Url.t -> unit
```

⟨*signature* Www.rem_active_cnx 93a⟩≡ (290b)
```
  val rem_active_cnx : Url.t -> unit
```

⟨*functions* Www.xxx_active_cnx 93b⟩≡ (291a) 93c ▷
```
  let is_active_cnx url =
    UrlSet.mem url !active_connexions
  let add_active_cnx url =
    active_connexions := UrlSet.add url !active_connexions
```

⟨*functions* Www.xxx_active_cnx 93c⟩+≡ (291a) ◁ 93b
```
  let rem_active_cnx url =
    active_connexions := UrlSet.remove url !active_connexions
```

7.4 Connections, Feed.t

⟨*type* Feed.internal 93d⟩≡ (285a 284b)
```
  (* An abstract notion of connection *)
  type internal = Unix.file_descr
```

⟨*type* Feed.t 93e⟩≡ (285a 284b)
```
  type t = {
    feed_read : bytes -> int -> int -> int;

    feed_schedule : (unit -> unit) -> unit;
    (* for abort? *)
    feed_unschedule : unit -> unit;

    feed_close : unit -> unit;

    (* for ?? *)
    feed_internal : internal
  }
```

⟨*signature* Feed.of_fd 93f⟩≡ (284b)
```
  val make_feed : Unix.file_descr -> (bytes -> int -> int -> int) -> t
```

XXX split and understand

⟨*function* Feed.of_fd 93g⟩≡ (285a)
```
  (* We should distinguish internal/external connections *)
  let make_feed (fd : Unix.file_descr) (do_read : bytes -> int -> int -> int) : t =

    let is_open = ref true in
    let action = ref None in
    let condition = Condition.create() in
    let first_read = ref false in

    (* ASSUMES: this is the first read on the fileevent *)
    let safe_read (buf : bytes) (offs : int) (len : int) : int =
      first_read := false;
      if !is_open
      then do_read buf offs len
      else 0
    in

    (* In other cases : this is non blocking but not fully threaded. *)
    let special_read (buf : bytes) (ofs : int) (len : int) : int =
       (* remove the normal handler *)
       Fileevent_.remove_fileinput fd;
       (* add a handler to trigger the condition *)
       Fileevent_.add_fileinput fd (fun () ->
         Fileevent_.remove_fileinput fd; (* remove myself *)
         Condition.set condition
       );
       (* wait for the condition to happen *)
       Condition.wait condition;
       (* Meanwhile, someone may have unscheduled/closed the
        * feed (e.g. abort). We call safe_read, but if the feed has been
        * closed, read will fail.
        * To know if we have to put back on schedule, check the *current*
        * state of action
        *)
       let n = try safe_read buf ofs len with _ -> 0 in
       (* reschedule; it is essential that Low.add_fileinput does not
        * call the event loop, otherwise we loose sequentiality of reads
        *)
       (match !action with
        | Some f ->
            Fileevent_.add_fileinput fd (fun () -> first_read := true; f())
        | None -> ()
        );
       (* and return *)
       n
    in
    { feed_read = (fun buf ofs len ->
        if !first_read
        then safe_read buf ofs len
        else special_read buf ofs len
     );

     feed_schedule = (fun f ->
        if not !is_open
        then Logs.err (fun m -> m "feed is closed, can't schedule")
        else
         (match !action with
          | Some _f -> (* we are already scheduled ! *)
              Logs.warn (fun  m -> m "feed already scheduled")
          | None ->
              action := Some f;
              Low.add_fileinput fd (fun () -> first_read := true; f())
          )
     );

     feed_unschedule = (fun () ->
       match !action with
       | Some _f ->
           Low.remove_fileinput fd;
           action := None
       | None ->
           (* this happens quite often (for all action codes which
            * do not process the body of the document, the feed got
            * unscheduled as the end of headers)
            *)
           ()
     );

     (* feed_close must be called only if the feed it *not* scheduled *)

     feed_close = (fun () ->
      (* if we abort during a state when we are waiting on the condition,
       * the feed is unscheduled but we never get out. So always change
       * the state
       *)
       Condition.set condition;
       if !is_open
       then
         (match !action with
         | Some _f -> Logs.err (fun m -> m "feed is scheduled, can't close")
         | None ->
             Unix.close fd;
             is_open := false;
             (* Condition.free condition RACE CONDITION HERE *)
         )
     );

     feed_internal = fd
    }
```

⟨*signature* Feed.internal 95a⟩≡ (284b)
```
  val internal : t -> internal
```

⟨*function* Feed.internal 95b⟩≡ (285a)
```
  let internal (feed : t) : internal =
    feed.feed_internal
```

7.5 Logging

⟨*signature* Document.dclose 95c⟩≡ (291b)
```
  val dclose : bool (* remactive *) -> handle -> unit
    (* [dclose remactive dh] closes a living dh *)
```

⟨*signature* Document.add_log 95d⟩≡ (291b)
```
  val add_log: handle -> string -> Www.aborter -> unit
```

⟨*signature* Document.put_log 95e⟩≡ (291b)
```
  val put_log : handle -> string -> unit
```

⟨*signature* Document.progress_log 95f⟩≡ (291b)
```
  val progress_log : handle -> int -> unit
```

⟨*signature* Document.end_log 95g⟩≡ (291b)
```
  val end_log : handle -> string -> unit
```

⟨*signature* Document.destroy_log 95h⟩≡ (291b)
```
  val destroy_log : handle -> bool -> unit
    (* logging functions *)
```

⟨*function* Document.dclose 95i⟩≡ (291c)
```
  (* Close a connexion. Should be called only by a fileinput callback
        or by somebody attempting to abort the connexion
     We remove the fd of the select before closing it since we don't want
     a spurious read to happen. This way we are somewhat independant of the
     Tk implementation
   *)

  let dclose (remactive : bool) (dh : handle) : unit =
    dh.document_feed.feed_unschedule();
    dh.document_feed.feed_close();
    if remactive then Www.rem_active_cnx dh.document_id.document_url
```

⟨*function* Document.add_log 96a⟩≡ (291c)
```
  let add_log (dh : handle) initmsg aborter =
    !add_log_backend dh initmsg aborter
```

⟨*function* Document.end_log 96b⟩≡ (291c)
```
  let end_log (dh : handle) (msg : string) : unit =
      dh.document_logger.logger_end msg;
      destroy_log dh true
```

⟨*functions* Document.xxx_log 96c⟩≡ (291c)
```
  let put_log (dh : handle) = dh.document_logger.logger_msg
  let destroy_log (dh : handle) = dh.document_logger.logger_destroy
  let progress_log (dh : handle) = dh.document_logger.logger_progress
```

7.6 Images

⟨*signature* Img.get 96d⟩≡ (320a)
```
  val get : <Cap.network; ..> ->
     Document.id -> Hyper.link -> (Url.t -> ImageData.t -> unit) ->
     Scheduler.progress_func -> unit
```

XXX ??

⟨*signature* Img.update 96e⟩≡ (320a)
```
  val update : < Cap.network; ..> ->
    Url.t -> unit
```

⟨*function* Img.get 96f⟩≡ (320c)
```
  (* ??? -> <> *)
  let get (caps : < Cap.network; ..>) (did : Document.id) (link : Hyper.link) cont
          (prog : Scheduler.progress_func) : unit =
    let wr = Www.make link in
    wr.www_headers <- "Accept: image/*" :: wr.www_headers;
    ImageScheduler.add_request (caps :> < Cap.network >) wr did cont prog
```

⟨*function* Img.update 96g⟩≡ (320c)
```
  (* ??? -> <> *)
  let update (caps : < Cap.network; ..>) (url : Url.t) : unit =
    try
      let (oldi, refs, headers) = ImageData.direct_cache_access url in
      let link = Hyper.default_link (Url.string_of url) in
      let wr = Www.make link in
      let date_received = Http_headers.get_header "date" headers in
      wr.www_headers <-
         ("If-Modified-Since: "^date_received)
         :: "Pragma: no-cache"
         :: wr.www_headers;

      ImageScheduler.add_request (caps :> < Cap.network >) wr (Document.DocumentIDSet.choose !refs)
        (fun _url i ->
          match oldi, i with
          | Still (ImagePhoto oldn) , Still (ImagePhoto newn) ->
            Imagephoto.copy oldn newn []
          | _, _ -> ()
        )
        Progress.no_meter

    with
      Not_found ->  (* either not in cache (bogus) or no date *)
        ()
```

7.7 Scheduling

⟨*toplevel comment* `Scheduler` 97a⟩≡ (323)
```
(*
 * Certain kind of documents need to be shared, such as in-lined images.
 * In this case, instead of working with Retrieve.f and the normal
 * document continuation, we queue the request to a scheduler, with a
 * continuation to be applied to an object representing the shared
 * information for that document.
 * E.G: for in-lined images, the shared information is the Tk-handle to
 * the image.
 *)
```

7.8 Progressing

⟨*type* `Scheduler.progress_func` 97b⟩≡ (324 323)
```
type progress_func = int option -> int -> unit
```

⟨*signature* `Progress.no_meter` 97c⟩≡ (318b)
```
val no_meter : Scheduler.progress_func
```

⟨*constant* `Progress.no_meter` 97d⟩≡ (318c)
```
let no_meter = (fun _ _ -> () : Scheduler.progress_func)
```

⟨*signature* `Progress.meter` 97e⟩≡ (318b)

⟨*function* `Progress.meter` 97f⟩≡ (318c)

Chapter 8

HTTP

8.1 The request

8.1.1 Http.req()

⟨*toplevel* Protos._2 98a⟩≡ (318a)
```
  let _ = Hashtbl.add protos Url.HTTP (Http.req, Cache.tobuffer)
  let _ = Hashtbl.add protos Url.HTTPS (Http.req, Cache.tobuffer)
```

⟨*signature* Http.req 98b⟩≡ (306c)
```
  val req: < Cap.network; ..> ->
    Www.request -> Document.continuation -> Www.aborter
```

⟨*function* Http.req 98c⟩≡ (306d)
```
  (* Wrappers returning the abort callback *)
  (* Retrieve.f -> <> (via protos) -> ... -> cont.document_process *)
  let req caps (wr : Www.request) (cont : Document.continuation) : Www.aborter =
    let cnx = request caps wr cont in
    (fun () -> cnx#abort)
```

⟨*function* Http.request 98d⟩≡ (306d)
```
  (* Issueing request, with the "retry" logic (unless is "always proxy" mode,
     we attempt first to connect directly to the host, and if it fails,
     we retry through the proxy
   *)
  (* Retrieve.f -> req (via protos) -> <> -> tcp_connect ->
   *  start_request (via contf) -> async_request -> process_response ->
   *   cont.document_process
   *)
  let request (caps : < Cap.network; ..>)
      (wr : Www.request) (cont : Document.continuation) : cnx =
    ⟨Http.request() if always proxy 221g⟩
    else
      let urlp = wr.www_url in
      if urlp.protocol = HTTP || urlp.protocol = HTTPS
      then
        let host =
          match urlp.host with
          | Some h -> h
          | _ -> raise (HTTP_error (s_ "Missing host in url"))
        in
        let port =
          match urlp.port with
          | Some p -> p
          | None ->
            (match urlp.protocol with
            | HTTP -> 80  (* default http port *)
            | HTTPS -> 443
            | _ -> raise (Impossible "can only have HTTP or HTTPS here")
            )
        in
        let is_https = urlp.protocol = HTTPS in
        try
          tcp_connect ~is_https caps host port wr.www_logging
              (fun cnx -> start_request false wr cont  cnx)
              (fun s aborted -> failed_request wr cont.document_finish s aborted)

        with HTTP_error _ -> (* direct failed, go through proxy *)
          ⟨Http.request() if http error on tcp_connect, try proxy 221h⟩
      else
        raise (HTTP_error (s_
               "INTERNAL ERROR\nHttp.request (not a distant http url): %s"
                 (Url.string_of wr.www_url)))
```

8.1.2 Http.tcp_connect()

⟨*function* Http.tcp_connect 99⟩≡ (306d)

```
  (* Open a TCP connection, asynchronously (except for DNS).
     We pass the continuation *)
  (* req | proxy_req -> request | proxy_request -> <> *)
  let tcp_connect ?(is_https = false) (caps : < Cap.network; ..>)
      (server_name : string) (port : int) logf contf errorf =

    (*  Find the inet address *)
    let server_addr : Unix.inet_addr =
      try Unix.inet_addr_of_string server_name
      with Failure _ ->
        ⟨Http.tcp_connect() if inet_add_of_string fails 101a⟩
    in

    (* Attempt to connect *)
    let sock = Unix.socket PF_INET SOCK_STREAM 0 in
    Unix.clear_nonblock sock;
    Unix.set_nonblock sock; (* set to non-blocking *)
    let cnx = new cnx (sock, errorf "User abort") in
    logf (s_ "Contacting host...");
    let ssl_handshake_and_continue cnx =
      if is_https then
        (try
          let s = Ssl.embed_socket cnx#fd ssl_ctx in
          Ssl.set_client_SNI_hostname s server_name;
          Ssl.connect s;
          cnx#set_ssl s;
          contf cnx
        with exn ->
          cnx#close;
          errorf (s_ "SSL handshake failed with %s: %s" server_name
                        (Printexc.to_string exn)) false)
      else
        contf cnx
    in
    try
      begin try
        CapUnix.connect caps sock (ADDR_INET(server_addr, port));
        (* just in case. Normally an error should be raised *)
        Unix.clear_nonblock sock; (* set to non-blocking *)
        logf (s_ "connection established");
        Logs.debug (fun m -> m "Connect returned without error !");

        (* because we need to return cnx *)
        Timer_.set 10 (fun () ->
          (* ! calling the continuation, e.g. start_request *)
          ssl_handshake_and_continue cnx
        );
        cnx
      with Unix.Unix_error((EINPROGRESS | EWOULDBLOCK | EAGAIN), "connect", _) ->
        ⟨Http.tcp_connect() if unix error when connect 101b⟩
      end
    with Unix.Unix_error(e,fn,_) ->  (* other errors in connect *)
      cnx#close;
      raise (HTTP_error (s_ "Cannot establish connection\n%s:%s"
                                 (Unix.error_message e) fn))
```

cnx

⟨*type* Http.status 100a⟩≡ (306d)

```
  (* Support for aborting requests while in connect/write/headers mode.
     When we start applying the document continuation, it is not our job
     anymore to abort the connection.
   *)
  type status =
    | Writing
    ⟨Http.status cases 105a⟩
```

⟨*class* Http.cnx 100b⟩≡ (306d)

```
  class cnx (sock, finish) =
   object (self)
    val mutable status = Writing
    val mutable fd = sock

    (* val finish = finish *)
    val mutable fdclosed = false  (* protect against double close *)
    val mutable aborted = false
    val mutable ssl_sock : Ssl.socket option = None

    method fd =
      fd
    method aborted =
      aborted
    method set_fd newfd =
      fd <- newfd
    method set_status s =
      status <- s
    method ssl_socket = ssl_sock
    method set_ssl s = ssl_sock <- Some s

    method write (buf : bytes) pos len =
      match ssl_sock with
      | None   -> Unix.write fd buf pos len
      | Some s -> Ssl.write s buf pos len

    method read (buf : bytes) pos len =
      match ssl_sock with
      | None   -> Low.read fd buf pos len
      | Some s -> Low.count_read (fun buf offs l ->
          try Ssl.read s buf offs l
          with Ssl.Read_error _ -> 0 (* treat all SSL read errors as EOF *)
        ) buf pos len

    method close =
      if not fdclosed then begin
        (match ssl_sock with
         | None -> ()
         | Some s -> (try Ssl.shutdown s with _ -> ()));
        Unix.close fd;
        fdclosed <- true
      end

    (* can be called from the aborter by the user or some exn handler *)
    method abort =
       if not aborted then begin
         aborted <- true;
         match status with
         | Writing ->
             Fileevent_.remove_fileoutput fd;
             self#close;
             finish true
         ⟨Http.cnx.abort() cases 106e⟩
      end
  end
```

Error management

⟨Http.tcp_connect() *if* inet_add_of_string *fails* 101a⟩≡ (99)
```
  try
    logf (s_ "Looking for %s ..." server_name);
    let adr = (Low.busy Munix.gethostbyname server_name).h_addr_list.(0) in
    logf (s_ "%s found" server_name);
    adr
  with Not_found ->
   raise (HTTP_error (s_ "Unknown host: %s" server_name))
```

⟨Http.tcp_connect() *if unix error when connect* 101b⟩≡ (99)
```
   (* that is ok, we are starting something *)
   let stuck = ref true in
   Fileevent_.add_fileoutput sock
     (* we are called when the cnx is established *)
     (fun () ->
       stuck := false;
       Fileevent_.remove_fileoutput sock;
       Unix.clear_nonblock sock; (* return to blocking mode *)
       begin try (* But has there been a cnx actually *)
         let _ = Unix.getpeername sock in
         logf (s_ "connection established");
         ssl_handshake_and_continue cnx
       with Unix.Unix_error(ENOTCONN, "getpeername", _) ->
         cnx#close;
         errorf (s_ "Connection refused to %s" server_name) false
        end
     );
    ⟨Http.tcp_connect() setup timeout 102b⟩
    cnx
```

Timout management

⟨*constant* Http.timeout 102a⟩≡ (306d)
```
  let timeout = ref 60  (* in seconds *)
```

⟨Http.tcp_connect() *setup timeout* 102b⟩≡ (101b)
```
  (* but also start the timer if nothing happens now
  * the kernel has a timeout, but it might be too long (linux)
  *)
  Timer_.set (1000 * !timeout)
    (fun () ->
       if not cnx#aborted && !stuck
       then begin
         Fileevent_.remove_fileoutput sock;
         cnx#close;
         errorf (s_ "Timeout during connect to %s" server_name) false
        end
    );
```

8.1.3 Http.start_request()

⟨*function* Http.start_request 102c⟩≡ (306d)
```
  let start_request (proxy_mode : bool) (wwwr : Www.request)
       (cont : Document.continuation) =
   fun (cnx : cnx) ->
    async_request proxy_mode wwwr
       (fun cnx -> process_response wwwr cont cnx) cnx
```

⟨*function* Http.async_request 102d⟩≡ (306d)
```
  (* Writing the request to the server
   *   TODO:  We might get some error here in write
   *   NOTE: tk doesn't allow two handles on the same fd, thus use CPS
   *         so that reading response is our continuation
   *)
  let async_request (proxy_mode : bool) (wwwr : Www.request) cont (cnx : cnx) =
    let b = Ebuffer.create 1024 in
    full_request (fun x -> Ebuffer.output_string b x) proxy_mode wwwr;
    let req = Ebuffer.get b in
    let len = Ebuffer.used b in
    let curpos = ref 0 in
    wwwr.www_logging (s_ "Writing request...");
    Fileevent_.add_fileoutput cnx#fd (fun _ ->
      let n = cnx#write (Bytes.of_string req) !curpos (len - !curpos) in (* blocking ? *)
      curpos := !curpos  + n;
      if !curpos = len then begin
        Fileevent_.remove_fileoutput cnx#fd;
        ⟨Http.async_request() log request string req if verbose 246a⟩
        (* ! calling the continuation, e.g. process_response *)
        cont cnx
      end)
```

8.1.4 Http.full_request()

⟨*function* Http.full_request 102e⟩≡ (306d)
```
  (* 'w' is the writer that will fill a buffer set in the caller *)
  (* request -> tcp_connect -> start_request (via cont) -> async_request -> <> *)
  let full_request (w : string -> unit) (proxy_mode : bool) (wwwr : Www.request) : unit =
    let url : string =
      ⟨Http.full_request() url value if proxy mode 221j⟩
      else Url.distant_path wwwr.www_url
    in
    ⟨Http.full_request() helper functions 103f⟩
    match wwwr.www_link.h_method with
    ⟨Http.full_request() method cases 103c⟩
```

⟨*signature* Url.distant_path 103a⟩≡ (286d)

```
  (* For http. The thing we have to send in the request *)
  val distant_path : t -> string
```

⟨*function* Url.distant_path 103b⟩≡ (287a)

```
  (* For http only *)
  let distant_path urlp =
    match urlp.path, urlp.search with
       None, None -> "/"
     | Some p, None -> "/"^p
     | Some p, Some s -> "/" ^ p ^ "?" ^ s
     | None, Some s -> "/?" ^ s (* ??? *)
```

GET

⟨Http.full_request() *method cases* 103c⟩≡ (102e) 104a ▷

```
  | GET ->
      w ("GET " ^ url ^ " HTTP/1.0\r\n");
      (* No General-Header *)
      w (std_request_headers());
      write_referer ();
      ⟨Http.full_request() write auth stuff 229f⟩
      write_other_headers();
      write_host();
      w "\r\n"
```

⟨*function* Http.std_request_headers 103d⟩≡ (306d)

```
  let std_request_headers() =
    Printf.sprintf "User-Agent: %s\r\n" !user_agent
```

⟨*constant* Http.user_agent 103e⟩≡ (306d)

```
  let user_agent =
    ref Version.http
```

⟨Http.full_request() *helper functions* 103f⟩≡ (102e) 103g ▷

```
  let write_other_headers () =
    wwwr.www_headers |> List.iter (fun s -> w s; w "\r\n");
    (* If no Accept given in request, write default one *)
    begin
      try
        Http_headers.get_header "accept" wwwr.www_headers |> ignore
      with Not_found -> w "Accept: */*\r\n"
    end
  in
```

⟨Http.full_request() *helper functions* 103g⟩+≡ (102e) ◁ 103f 228 ▷

```
  (* Host: header for virtual domains *)
  let write_host () =
    match wwwr.www_url.host with
    | None -> (* never happens *) ()
    | Some h ->
        (match wwwr.www_url.port with
        | None -> w ("Host: "^h^"\r\n")
        | Some p ->  w ("Host: "^h^":"^string_of_int p^"\r\n")
        )
  in
```

POST

⟨Http.full_request() *method cases* 104a⟩+≡ (102e) ◁103c 104c▷

```
  | POST data ->
      w ("POST "^url^" HTTP/1.0\r\n");
      (* No General-Header *)
      w (std_request_headers());
      write_referer ();
      ⟨Http.full_request() write auth stuff 229f⟩
      write_other_headers();
      write_host();
      (* 8.2.1 *)
      w ("Content-Type: application/x-www-form-urlencoded\r\n");
      (* 7.2 note *)
      w ("Content-Length: " ^ string_of_int (String.length data)^ "\r\n");
      w "\r\n";
      w data
```

HEAD

⟨Hyper.link_method *other cases* 104b⟩≡ (18c)

```
  | HEAD
```

⟨Http.full_request() *method cases* 104c⟩+≡ (102e) ◁104a

```
  | HEAD ->
      w ("HEAD "^url^" HTTP/1.0\r\n");
      (* No General-Header *)
      w (std_request_headers());
      write_referer ();
      ⟨Http.full_request() write auth stuff 229f⟩
      write_other_headers();
      write_host();
      w "\r\n"
```

8.1.5 Http.failed_request()

⟨*function* Http.failed_request 104d⟩≡ (306d)

```
  (* shared error *)
  let failed_request (wr : Www.request) finish =
   fun s aborted ->
    finish aborted;
    Www.rem_active_cnx wr.www_url;
    wr.www_logging (s_ "Failed");
    wr.www_error#f (s_ "Request for %s failed\n%s" (Url.string_of wr.www_url) s)
```

8.2 The response

8.2.1 Http.process_response()

⟨*function* Http.process_response 104e⟩≡ (306d)

```
  (* Read headers and run continuation *)
  (* tcp_connect -> <> (via contf) *)
  let process_response (wwwr : Www.request) (cont : Document.continuation) =
   fun (cnx : cnx) ->
    let url : string = Url.string_of wwwr.www_url in
    wwwr.www_logging (s_ "Reading headers...");

    let dh =
      Document.{ document_id = document_id wwwr;
        document_referer = wwwr.www_link.h_context;
        document_fragment = wwwr.www_fragment;

        document_status = 0;
        dh_headers = [];
        document_feed = Feed.make_feed cnx#fd cnx#read;

        document_logger = Document.tty_logger;
      }
    in
    cnx#set_status (Reading dh);

    let stuck = ref true in
    (* set up a timer to abort if server is too far/slow *)
    ⟨Http.process_response() setup a timer 107b⟩

    (* reading the headers *)
    dh.document_feed.feed_schedule
      (fun () ->
         stuck := false;
         ⟨Http.process_response() reading headers 105c⟩
```

⟨Http.status *cases* 105a⟩≡ (100a) 106d ▷

```
  | Reading of Document.handle
```

8.2.2 Reading headers

⟨Document.handle *other fields* 105b⟩≡ (22b) 199a ▷

```
  mutable document_status : int;
    (* Status code of response *)
```

⟨Http.process_response() *reading headers* 105c⟩≡ (104e)

```
  try
    if dh.dh_headers = [] then begin
      (* it should be the HTTP Status-Line *)
      let l = Low.read_line_fn cnx#read in
      dh.document_status <- (Http_headers.parse_status l).status_code;
      dh.dh_headers <- [l] (* keep it there *)
    end else
       dh.dh_headers <- read_headers cnx#read dh.dh_headers
  with
  (* each branch must unschedule *)
  ⟨Http.process_response() feed schedule callback failure cases 106c⟩
```

⟨*exception* Http.End_of_headers 105d⟩≡ (306)

```
  exception End_of_headers
```

⟨*signature* Http.read_headers 106a⟩≡ (306c)
```
  (* [read_headers fd]
   *  reads HTTP headers from a fd
   *    raises End_of_file
   *    raises Invalid_HTTP_header
   *)
  val read_headers:
    (bytes -> int -> int -> int) -> string list -> string list
```

⟨*function* Http.read_headers 106b⟩≡ (306d)
```
  let read_headers read_fn previous =
    let l = Low.read_line_fn read_fn in
     if String.length l = 0
     then raise End_of_headers (* end of headers *)
     else
       if l.[0] = ' ' || l.[0] = '\t'
       then  (* continuation *)
         match previous with
         | [] -> raise (Http_headers.Invalid_header ("invalid continuation " ^ l))
         | s :: rest -> (s^l) :: rest
        else l :: previous
```

⟨Http.process_response() *feed schedule callback failure cases* 106c⟩≡ (105c) 106f▷
```
  | End_of_headers ->
      dh.document_feed.feed_unschedule();
      cnx#set_status Discharged;

      (* ! call the continuation, finally! *)
      cont.document_process dh
```

⟨Http.status *cases* 106d⟩+≡ (100a) ◁105a
```
  | Discharged
```

⟨Http.cnx.abort() *cases* 106e⟩≡ (100b) 107a▷
```
  | Discharged -> ()
```

8.2.3 Error management

⟨Http.process_response() *feed schedule callback failure cases* 106f⟩+≡ (105c) ◁106c 106g▷
```
  | Not_found -> (* that's what parse_status raises. HTTP/0.9 dammit *)
      failwith "HTTP 0.9 not handled anymore"
```

⟨Http.process_response() *feed schedule callback failure cases* 106g⟩+≡ (105c) ◁106f 106h▷
```
  | Unix.Unix_error(e,_,_) ->
      cnx#abort;
      wwwr.www_error#f (s_ "Error while reading headers of %s\n%s" url
                             (Unix.error_message e))
```

⟨Http.process_response() *feed schedule callback failure cases* 106h⟩+≡ (105c) ◁106g 106i▷
```
  | Http_headers.Invalid_header s ->
      cnx#abort;
      wwwr.www_error#f (s_ "Error while reading headers of %s\n%s" url s)
```

⟨Http.process_response() *feed schedule callback failure cases* 106i⟩+≡ (105c) ◁106h
```
  | End_of_file ->
      cnx#abort;
      wwwr.www_error#f (s_ "Error while reading headers of %s\n%s" url "eof"))
```

⟨Http.cnx.abort() *cases* 107a⟩+≡ (100b) ◁106e
```
  | Reading dh ->
      Document.dclose true dh;
      finish true
```

8.2.4 Timeout management

⟨Http.process_response() *setup a timer* 107b⟩≡ (104e)
```
  let rec timout () =
     Timer_.set (1000 * !timeout)
      (fun () ->
         if not cnx#aborted && !stuck
         then
           match
            wwwr.www_error#ari (s_ "Timeout while waiting for headers of %s" url)
           with
           (* TODO: use proper enum for www_error#ari return type *)
           | 0 -> (* abort *) if !stuck then cnx#abort
           | 1 -> (* retry *) timout ()
           | 2 -> (* ignore *) ()
           | _ -> ()
    )
  in

  timout();
```

8.3 Headers merging

⟨*signature* Http_headers.merge_headers 107c⟩≡ (300g)
```
  val merge_headers : Messages.header list -> Messages.header list -> Messages.header list
    (* [merge_headers oldhs newhs] merges headers, overriding headers in
       [oldhs] by headers in [newhs] *)
```

⟨*function* Http_headers.merge_headers 107d⟩≡ (302)
```
  (* Keep only unmodified headers *)
  let merge_headers oldh newh =
    let rec filter acc = function
        [] -> acc
     | s::l ->
         if String.length s > 5 && String.sub s 0 5 = "HTTP/" then
          filter acc l
         else
          try
           let t = header_type s in
           let _d = get_header t newh in
         filter acc l
          with
             Invalid_header _ ->
             Log.debug (sprintf "Dumping invalid header (%s)" s);
                 filter acc l
           | Not_found -> filter (s::acc) l in
    (filter [] oldh) @ newh
```

⟨*signature* Http_headers.header_type 107e⟩≡ (300g)
```
  val header_type : string -> string
    (* [header_type h] returns the field_name token of [h], in lowercase *)
```

⟨*function* Http_headers.header_type 108a⟩≡ (302)
```
  let header_type s =
    match Str.bounded_split (regexp "[:]") s 2 with
    | [t;_] -> String.lowercase_ascii t
    | _ -> raise (Invalid_header s)
```

⟨*signature* Http_headers.remove_headers 108b⟩≡ (300g)
```
  val remove_headers : Messages.header list -> string list -> Messages.header list
    (* [remove_headers hs field_names] returns [hs] without the headers
       with field_name present in [field_names] *)
```

⟨*function* Http_headers.remove_headers 108c⟩≡ (302)
```
  let remove_headers hs names =
    Log.debug "remove headers";
    let rec rem acc = function
        [] -> acc
     | h::l ->
         try
          let t = header_type h in
            if List.mem t names then rem acc l
            else rem (h::acc) l
          with Invalid_header s ->
             Log.debug (sprintf "Dumping invalid header (%s)" s);
             rem acc l

     in rem [] hs
```

8.4 Time

⟨*type* Http_date.http_time 108d⟩≡ (299)
```
  (* Based on Unix.tm *)
  type http_time =
    { ht_sec : int;                     (* Seconds 0..59 *)
      ht_min : int;                     (* Minutes 0..59 *)
      ht_hour : int;                    (* Hours 0..23 *)
      ht_mday : int;                    (* Day of month 1..31 *)
      ht_mon : int;                     (* Month of year 0..11 *)
      ht_year : int;                    (* Year - 1900 *)
      ht_wday : int }                   (* Day of week (Sunday is 0) *)
```

⟨*signature* Http_date.expired 108e⟩≡ (299a)
```
  val expired : http_time -> bool
    (* Determines if an http_time is in the past *)
```

⟨*signature* Http_date.compare 108f⟩≡ (299a)
```
  val compare : http_time -> http_time -> int
    (* Compares two http_times *)
```

⟨*signature* Http_date.string_of_ht 108g⟩≡ (299a)
```
  val string_of_ht : http_time -> string
    (* Text version (RFC822) of an http time stamp *)
```

⟨*signature* Http_date.tm_of_ht 108h⟩≡ (299a)
```
  val tm_of_ht : http_time -> Unix.tm
```

⟨*signature* Http_date.stamp_of_ht 108i⟩≡ (299a)
```
  val stamp_of_ht : http_time -> float
```

⟨*signature* Http_date.ht_of_stamp 109a⟩≡ (299a)
```
  val ht_of_stamp : float -> http_time
```

⟨*function* Http_date.expired 109b⟩≡ (299b)
```
  let expired ht =
    let now = gmtime(time()) in
    let lht =
      [ht.ht_year; ht.ht_mon; ht.ht_mday; ht.ht_hour; ht.ht_min; ht.ht_sec]
    and lnow =
      [now.tm_year; now.tm_mon; now.tm_mday; now.tm_hour; now.tm_min; now.tm_sec]
    in
      compare_time (lht, lnow) <= 0
```

⟨*function* Http_date.compare 109c⟩≡ (299b)
```
  let compare ht1 ht2 =
   compare_time
    ([ht1.ht_year; ht1.ht_mon; ht1.ht_mday; ht1.ht_hour; ht1.ht_min; ht1.ht_sec],
     [ht2.ht_year; ht2.ht_mon; ht2.ht_mday; ht2.ht_hour; ht2.ht_min; ht2.ht_sec])
```

⟨*function* Http_date.string_of_ht 109d⟩≡ (299b)
```
  let string_of_ht ht =
    sprintf "%s, %02d %s %d %02d:%02d:%02d GMT"
        (asc_wkday ht.ht_wday)
        ht.ht_mday
        (asc_month ht.ht_mon)
        (ht.ht_year + 1900)
        ht.ht_hour
        ht.ht_min
        ht.ht_sec
```

⟨*function* Http_date.tm_of_ht 109e⟩≡ (299b)
```
  (*
  let has_dst = localtime(time()).tm_isdst
  *)
  let tm_of_ht ht = {
      tm_sec = ht.ht_sec;
      tm_min = ht.ht_min;
      tm_hour = ht.ht_hour;
      tm_mday = ht.ht_mday;
      tm_mon = ht.ht_mon;
      tm_year = ht.ht_year;
      tm_wday = ht.ht_wday;
      tm_yday = 0;
      tm_isdst = false        (* I don't have a clue here *)
     }
```

⟨*function* Http_date.stamp_of_ht 109f⟩≡ (299b)
```
  let stamp_of_ht ht =
     fst (mktime (tm_of_ht ht))
```

⟨*function* Http_date.ht_of_stamp 109g⟩≡ (299b)
```
  let ht_of_stamp ut =
    let tm = gmtime ut in {
      ht_sec = tm.tm_sec;
      ht_min = tm.tm_min;
      ht_hour = tm.tm_hour;
      ht_mday = tm.tm_mday;
      ht_mon = tm.tm_mon;
      ht_year = tm.tm_year;
      ht_wday = tm.tm_wday
       }
```

8.5 HTTPS and TLS and SSL

Chapter 9

Viewers

⟨*signature* Viewers.view 111a⟩≡ (339c)
```
  val f : Widget.widget -> context -> Document.handle -> display_info option
```

9.1 Viewers.f()

⟨*function* Viewers.view 111b⟩≡ (340c)
```
  (* the meat (was called view) *)
  (* Nav.absolutegoto -> Nav.request -> Nav.process_viewer (via process) -> <>
   *   -> Plain.viewer | Htmlw.viewer (via viewers)
   *)
  and f frame (ctx : context) (dh : Document.handle) : display_info option =
    try
      let ctype = Http_headers.contenttype dh.dh_headers in
      let (typ, sub), pars = Lexheaders.media_type ctype in
      try (* Get the viewer *)
        Logs.debug (fun m -> m "Viewers.view %s/%s" typ sub);
        let viewer =
          try Hashtbl.find viewers (typ,sub)
          with Not_found ->
            Logs.warn (fun m -> m "didn't find viewer for %s/%s" typ sub);
            Hashtbl.find viewers (typ,"*")
        in
        match viewer with
        ⟨Viewers.view match viewer cases 112g⟩
      with
      ⟨Viewers.view exn handler 1 112h⟩
    with
    ⟨Viewers.view exn handler 2 112i⟩
```

⟨*signature* Viewers.add_viewer 111c⟩≡ (339c)
```
  val add_viewer : Http_headers.media_type -> t -> unit
      (* [add_viewer type viewer] *)
```

⟨*function* Viewers.add_viewer 111d⟩≡ (340c)
```
  (* That's for internal viewers only *)
  let add_viewer ctype (viewer : t) =
    Logs.info (fun m -> m "adding viewer for %s/%s" (fst ctype) (snd ctype));
    Hashtbl.add viewers ctype (Internal viewer)
```

⟨*signature* Viewers.rem_viewer 111e⟩≡ (339c)
```
  val rem_viewer : Http_headers.media_type -> unit
```

⟨*function* Viewers.rem_viewer 112a⟩≡ (340c)
```
  let rem_viewer ctype =
    Hashtbl.remove viewers ctype
```

⟨*constant* Viewers.builtin_viewers 112b⟩≡ (340c)
```
  let builtin_viewers = ref []
```

⟨*signature* Viewers.add_builtin 112c⟩≡ (339c)
```
  val add_builtin : Http_headers.media_type -> t -> unit
      (* [add_builtin type viewer] makes viewer a builtin for type *)
```

⟨*function* Viewers.add_builtin 112d⟩≡ (340c)
```
  let add_builtin t v =
    (* This will not work because add_builtin is usually called
     * for toplevel vars like let _ = add_builtin ... in plain.ml
     * at the moment where Logs are not yet enabled
   Logs.info (fun m -> m "adding builtin viewer for %s/%s"
            (fst t) (snd t));
    *)
    builtin_viewers := (t,v) :: !builtin_viewers
```

⟨*signature* Viewers.reset 112e⟩≡ (339c)
```
  val reset : unit -> unit
```

⟨*function* Viewers.reset 112f⟩≡ (340c)
```
  (* ??? -> <> *)
  let reset () =
    Logs.info (fun m -> m "resetting viewers");

    (* Reset the viewer table *)
    Hashtbl.clear viewers;

    (* Restore the builtin viewers *)
    List.iter (fun (x,y) -> add_viewer x y) !builtin_viewers;

    ⟨Viewers.reset() setting other viewers 118d⟩
    ()
```

9.1.1 Dispatching

⟨Viewers.view *match viewer cases* 112g⟩≡ (111b) 119a ▷
```
  | Internal viewer ->
      ctx#log (s_ "Displaying...");
      viewer pars frame ctx (Decoders.insert dh)
```

9.1.2 Error management

⟨Viewers.view *exn handler 1* 112h⟩≡ (111b) 120c ▷
```
  | Failure "too late" -> (* custom for our internal viewers *)
      Document.dclose true dh;
      Document.destroy_log dh false;
      None
```

⟨Viewers.view *exn handler 2* 112i⟩≡ (111b)
```
  | Http_headers.Invalid_header e ->
      ctx#log (s_ "Malformed type: %s" e);
      unknown frame ctx dh
  | Not_found ->
      (* Content-type was not defined in the headers *)
      (* and could not be computed from url *)
      unknown frame ctx dh
```

⟨*function* Viewers.unknown 113a⟩≡ (340c)
```
  let rec unknown (frame : Widget.widget) (ctx : context) (dh : Document.handle)
     : display_info option =
    match Frx_dialog.f frame (Mstring.gensym "error")
           (s_ "MMM Warning")
           (s_ "No MIME type given for the document\n%s"
                 (Url.string_of dh.document_id.document_url))
           (Tk.Predefined "question")
           0
           [s_ "Retry with type"; s_ "Save to file"; s_ "Abort"]
    with
    | 0 ->
      let v = Textvariable.create_temporary frame in
      Textvariable.set v "text/html";
      if Frx_req.open_simple_synchronous (s_ "MIME type") v then
         let ctype = Textvariable.get v in
         dh.dh_headers <- ("Content-Type: " ^ ctype) :: dh.dh_headers;
         (* try again *)
         f frame ctx dh
      else begin
         Save.interactive (fun _ -> ()) dh;
         None
      end
   | 1 -> Save.interactive (fun _ -> ()) dh; None
   | 2 -> Document.dclose true dh; None
   | _ -> assert false (* property of dialogs *)
```

9.2 Content types

9.2.1 text/plain, Plain.viewer()

⟨*toplevel* Plain._1 113b⟩≡ (342a)
```
  let _ =
    Viewers.add_builtin ("text","plain") viewer
```

⟨*function* Plain.display_plain 113c⟩≡ (342a)
```
  (* Viewing text/plain *)
  (* old: was called display_plain *)
  (* Viewer.f -> <> (via Viewers.viewers) *)
  let viewer : Viewers.t =
   fun _mediapars
      (top : Widget.widget) (vcontext : Viewers.context) (dh : Document.handle) :
       Viewers.display_info option ->
    let disp = new display_plain (top,vcontext,dh) in
    disp#init;
    Some (disp :> Viewers.display_info)
```

⟨*class* Plain.plain 113d⟩≡ (342a)
```
  class display_plain ((top : Widget.widget),
                       (ctx : Viewers.context),
                       (dh : Document.handle)) =
   object (self)
    inherit Viewers.display_info () as _di  (* gives us basic features *)
  (*
    inherit Htmlw.viewer_globs (ctx, dh)
  *)
    method ctx = ctx
    method di_title =
      Url.string_of dh.document_id.document_url

    ⟨Plain.plain private fields 114b⟩

    ⟨Plain.plain frame widget methods 114a⟩
    ⟨Plain.plain init method 114c⟩
    ⟨Plain.plain adding text method 115b⟩

    ⟨Plain.plain progress methods 205a⟩
    ⟨Plain.plain abort methods 203d⟩
    ⟨Plain.plain redisplay methods 204e⟩

    ⟨Plain.plain graphic cache destroy methods 238b⟩
    ⟨Plain.plain empty methods 164e⟩
    ⟨Plain.plain other methods or fields 192d⟩
  end
```

```
⟨Plain.plain frame widget methods 114a⟩≡                                   (113d)
  val frame =
    if not (Winfo.exists top)
    then failwith "too late"
    else Frame.create top [Class "Plain"]
  method frame = frame

  method di_widget = frame
```

```
⟨Plain.plain private fields 114b⟩≡                                  (113d)  203c ▷
  (* text widget, will be set to the proper value in init() *)
  val mutable (*private*) tw = Widget.default_toplevel
```

```
⟨Plain.plain init method 114c⟩≡                                            (113d)
  method init =
    Logs.debug (fun m -> m "Plain#init");
    ⟨Plain.plain#init set header widgets 115c⟩

    (* Scrollable text widget *)
    let hgroup = Frame.create_named frame "textw" [Class "Plain"] in
    let ftext, text =
      Frx_text.new_scrollable_text hgroup [Wrap WrapWord; State Disabled] true
    in
    ⟨Plain.plain#init tk fixes on text widget 115d⟩
    (* IN THIS ORDER -- RESIZING *)
    pack [ftext][Side Side_Left; Fill Fill_Both; Expand true];
    pack [hgroup][Fill Fill_Both; Expand true];

    ⟨Plain.plain#init setup fonts 155d⟩

    tw <- text;

    ⟨Plain.plain#init locals 114d⟩
    dh.document_feed.feed_schedule (fun () ->
      ⟨Plain.plain#init feed schedule callback 115a⟩
     );
```

```
⟨Plain.plain#init locals 114d⟩≡                                            (114c)
  let buffer = Bytes.create 2048 in
  let size =
    try Some (Http_headers.contentlength dh.dh_headers)
    with Not_found -> None (* duh *)
  in
  let read = ref 0 in
  let lastwascr = ref false in
```

⟨Plain.plain#init *feed schedule callback* 115a⟩≡ (114c)

```
  try
    let n = dh.document_feed.feed_read buffer 0 2048 in
    if n = 0 then begin
      if !lastwascr
      then self#add_text "\n";
      self#add_text ""; (* special case to indicate end *)
      self#set_progress (Some !read) !read;
      self#finish false (* not abort *)
    end else begin
      read := !read + n;
      self#set_progress size !read;
      let s,flag = Mstring.norm_crlf !lastwascr (Bytes.to_string buffer) 0 n in
      lastwascr := flag;
      self#add_text s
    end
  with Unix.Unix_error(_,_,_) ->
     self#set_progress size (-1);
     self#di_abort
```

⟨Plain.plain *adding text method* 115b⟩≡ (113d)

```
  val mutable pending = true
  method add_text s =
    if s = ""
    then pending <- false
    else
      if Winfo.exists tw then begin
        Text.configure tw [State Normal];
        Text.insert tw Frx_text.textEnd s [];
        Text.configure tw [State Disabled]
      end
```

⟨Plain.plain#init *set header widgets* 115c⟩≡ (114c)

```
  (*
  let hgbas, progf = Htmlw.progress_report frame ctx in
  set_progress <- progf;
  pack [hgbas] [Side Side_Bottom; Fill Fill_X];
  let (headgroup,_,_,_,_) =
    Htmlw.html_head_ui dh.document_headers (fun () -> ()) (ref false)
     frame ctx
  in
  pack [headgroup][Side Side_Top; Fill Fill_X];
  *)
```

⟨Plain.plain#init *tk fixes on text widget* 115d⟩≡ (114c)

```
  (* Tk4.0pl3 fix, + avoid cb to scrollbar *)
  Text.configure text [TakeFocus true; InsertOffTime 0];
  Frx_text.addsearch text;
```

9.2.2 text/html, Htmlw.viewer()

⟨*toplevel* Htmlw._1 115e⟩≡ (391b)

```
  let _ =
    Viewers.add_builtin ("text","html") viewer
```

⟨*function* Htmlw.display_html 116a⟩≡ (391b)
```
(* Nav.absolutegoto -> Nav.request -> Nav.process_viewer (via process) ->
 *  Viewers.f -> <> (via viewers)
 *)
let viewer : Viewers.t =
 fun mediapars top (ctx : Viewers.context) (dh : Document.handle) ->
  let imgmanager = Imgload.create() in
  let disp = new display_html (top,ctx,mediapars,imgmanager,dh) in
  disp#init true;
  Some (disp :> Viewers.display_info)
```

9.3 MIME types

⟨*type* Http_headers.hint 116b⟩≡ (302 300g)
```
(* Associating MIME type or Content-Encoding with file/URI suffix *)
type hint =
  | ContentType     of Messages.header
  | ContentEncoding of Messages.header
```

⟨*constant* Http_headers.default_hints 116c⟩≡ (302)
```
let default_hints = [
  "html", ContentType  "Content-Type: text/html";
  "htm", ContentType  "Content-Type: text/html";

  "txt",   ContentType  "Content-Type: text/plain";

  "ps",   ContentType  "Content-Type: application/postscript";
  "dvi",   ContentType  "Content-Type: application/x-dvi";

  "gif", ContentType  "Content-Type: image/gif";
  "jpeg", ContentType  "Content-Type: image/jpeg";
  "jpg", ContentType  "Content-Type: image/jpeg";
  "tiff", ContentType  "Content-Type: image/tiff";
  "tif", ContentType  "Content-Type: image/tiff";

  "au",  ContentType  "Content-Type: audio/basic";
  "snd", ContentType  "Content-Type: audio/basic";
  "wav", ContentType  "Content-Type: audio/x-wav";
  "mid", ContentType  "Content-Type: audio/midi";

  "mpeg", ContentType  "Content-Type: video/mpeg";
  "mpg", ContentType  "Content-Type: video/mpeg";
  "avi", ContentType  "Content-Type: video/avi";
  "fli", ContentType  "Content-Type: video/fli";
  "flc", ContentType  "Content-Type: video/fli";

   ⟨Http_headers.suffixes elements 176c⟩
]
```

9.3.1 Http_headers.suffixes

⟨*constant* Http_headers.suffixes 116d⟩≡ (302)
```
let suffixes =
   (Hashtbl.create 101 : (string, hint) Hashtbl.t)
```

⟨*toplevel* Http_headers._1 117a⟩≡ (302)
```
  (* Even if we don't have a suffix file... *)
  (* If the suffix file says otherwise, it will have priority *)
  let _ =
    default_hints |> List.iter (fun (s,t) -> Hashtbl.add suffixes s t)
```

9.3.2 mmm -suffixes

⟨Main.main() *locals* 117b⟩+≡ (29c) ◁30e 188g▷
```
  let sufxfile = ref (Mmm.user_file "mime.types") in
```

⟨Main.main() *command line options* 117c⟩+≡ (29c) ◁31b 188h▷
```
  "-suffixes", Arg.String (fun s -> sufxfile := (Fpath.v s)),
  " <file> Suffix file";
```

⟨Main.main() *suffix initialisation* 117d⟩≡ (30c)
```
  (* Suffix mapping to Content-Type and Content-Encoding *)
  if Sys.file_exists !!(!sufxfile)
  then Http_headers.read_suffix_file !!(!sufxfile);
```

⟨*signature* Http_headers.read_suffix_file 117e⟩≡ (300g)
```
  val read_suffix_file : string -> unit
```

⟨*function* Http_headers.read_suffix_file 117f⟩≡ (302)
```
  (* In the file, we select ContentType if there is a slash,
     ContentEncoding otherwise *)
  let read_suffix_file f =
   try
     let ic = open_in f in
     try while true do
       let l = input_line ic in
       if l <> "" && l.[0] <> '#'
       then
         let tokens =
           split_str (function ' '|'\t' -> true | _ -> false) l in
         match tokens with
         | [] -> ()
         | x::l ->
             try
               let _ = String.index x '/' in
               l |> List.iter  (fun sufx ->
                  Hashtbl.add suffixes sufx  (ContentType ("Content-Type: "^x))
               )
            with Not_found ->
              l |> List.iter  (fun sufx ->
                  Hashtbl.add suffixes sufx
                    (ContentEncoding ("Content-Encoding: "^x))
              )
       done
     with End_of_file -> close_in ic
   with Sys_error _ ->  ()
```

9.3.3 Filename suffix to content type

⟨*function* Http_headers.hints 118a⟩≡ (302)
```
  let hints path =
    (* Get the url suffix *)
    let sufx = Mstring.get_suffix path in
    try
      let v =
        try Hashtbl.find suffixes sufx
        with Not_found ->
          Hashtbl.find suffixes (String.lowercase_ascii sufx)
      in
      match v with
      | ContentType t -> [t] (* good, we have a type *)
      | ContentEncoding e ->
         (* we have an encoding, but do we have a type too ? *)
         let path2 = Filename.chop_suffix path ("."^sufx) in
         let sufx2 = Mstring.get_suffix path2 in
         begin try
           let v2 = Hashtbl.find suffixes sufx2 in
           match v2 with
           | ContentType t -> (* good, we have a type *)
              [t;e]
           | ContentEncoding _ -> [e] (* nah, forget it *)
          with Not_found -> [e] (* no type *)
        end
    with Not_found -> [] (* no hint ... *)
```

9.3.4 Retype.f()

⟨*signature* Retype.f 118b⟩≡ (306a)
```
  val f : Document.handle -> unit
    (* physically modify the headers, adding ContentType/ContentEncoding
     * from URL suffixes if this information is missing from the headers.
     *)
```

⟨*function* Retype.f 118c⟩≡ (306b)
```
  (* Attempt to find a decent Content-Type *)
  let f dh =
    let url = Url.string_of dh.document_id.document_url in
    try
      let ctype = Http_headers.contenttype dh.dh_headers in
      let mtyp,_pars = Lexheaders.media_type ctype in
      if mtyp = ("application","octet-stream")
      then dh.dh_headers <- merge_headers dh.dh_headers (hints url)
    with Not_found ->
      let hints = Http_headers.hints url in
      dh.dh_headers <- merge_headers dh.dh_headers hints
```

9.4 External viewer

⟨Viewers.reset() *setting other viewers* 118d⟩≡ (112f) 121b ▷
```
  (* Preference settings *)
  Tkresource.stringlist "externalViewers" [] |> List.iter (fun ctype ->
    try
      let (typ,sub), _pars = Lexheaders.media_type ctype in
      Hashtbl.add viewers (typ,sub) External
    with Http_headers.Invalid_header e ->
      Error.f (s_ "Invalid MIME type %s\n%s" ctype e)
  );
```

⟨Viewers.view *match viewer cases* 119a⟩+≡ (111b) ◁112g 120b▷

```
  | External ->
      ctx#log (s_ "Displaying externally");
      extern (Decoders.insert dh) (sprintf "%s/%s" typ sub);
      None
```

⟨*function* Viewers.extern 119b⟩≡ (340c)

```
  (* "interactive" version:
   *    send data to metamail as it arrives, but allow abort
   * NOTE: There are sometimes weird errors when the child dumps core
   *       between fork/exec with no apparent reason (on SunOS4.1 only)
   *)
  let extern (dh : Document.handle) (ctype : string) : unit =
    let (pin, pout) = Unix.pipe() in
    (* children must not keep pout open *)
    Unix.set_close_on_exec pout;
    match Low.fork() with
    | 0 ->
        Unix.dup2 pin Unix.stdin;
        Unix.close pin;
        Munix.execvp "metamail" [| "metamail"; "-b"; "-x"; "-c"; ctype |]
    | pid ->
        Unix.close pin;
        let kill () =
           try Unix.kill pid 2
           with Unix.Unix_error (e,_,_) ->
            Log.f (sprintf "Can't kill child (%s)" (Unix.error_message e))
        in
        let url = Url.string_of dh.document_id.document_url in
        Document.add_log dh
          (s_ "Retrieving %s\nfor external display with MIME type %s" url ctype)
          kill;

        let red = ref 0 in
        let size =
          try Http_headers.contentlength dh.dh_headers
          with Not_found -> 40000 (* duh *)
        in
        let buffer = Bytes.create 4096 in
        dh.document_feed.feed_schedule
          (fun () ->
            try
              let n = dh.document_feed.feed_read buffer 0 4096 in
              if n = 0 then begin
                Document.dclose true dh;
                Unix.close pout;
                Document.end_log dh (s_ "End of transmission")
              end else begin
                ignore (Unix.write pout buffer 0 n);
                red := !red + n;
                Document.progress_log dh (!red * 100 / size)
              end
            with Unix.Unix_error(e,_,_) ->
              Log.f (sprintf "Error writing to viewer (%s)"
                      (Unix.error_message e));
              Document.dclose true dh;
              kill();
              Unix.close pout;
              Document.destroy_log dh false;
              Error.f (s_ "Error during retrieval of %s" url)
         )
```

9.5 Interactive viewer

⟨Viewers.spec *other cases* 120a⟩≡ (23h) 121a▷

```
  (*| Interactive  (* ask what to do about it *)*)
```

⟨Viewers.view *match viewer cases* 120b⟩+≡ (111b) ◁119a 121c▷

```
  (*| Interactive ->
      interactive frame ctx dh ctype*)
```

⟨Viewers.view *exn handler 1* 120c⟩+≡ (111b) ◁112h

```
  | Not_found ->
     (* we don't know how to handle this *)
     ctx#log (s_ "Displaying externally");
     interactive frame ctx dh ctype
```

⟨*function* Viewers.interactive 120d⟩≡ (340c)

```
  and interactive frame (ctx : context) (dh : Document.handle) (ctype : string)
     : display_info option =
    match Frx_dialog.f frame (Mstring.gensym "error")
           (s_ "MMM Viewers")
           (s_
            "No behavior specified for MIME type\n%s\ngiven for the document\n%s"
             ctype
             (Url.string_of dh.document_id.document_url))
           (Tk.Predefined "question")
           0
           [s_ "Retry with another type";
            s_ "Display with metamail";
            s_ "Save to file";
            s_ "Abort"]
    with
    | 0 ->
        let v = Textvariable.create_temporary frame in
        Textvariable.set v "text/html";
        if Frx_req.open_simple_synchronous (s_ "MIME type") v then
          let ctype = Textvariable.get v
          in
          dh.dh_headers <- ("Content-Type: " ^ ctype) :: dh.dh_headers;
          (* try again *)
          f frame ctx dh
        else begin
          Save.interactive (fun _ -> ()) dh;
          None
        end
    | 1 -> extern (Decoders.insert dh) ctype; None
    | 2 -> Save.interactive (fun _ -> ()) dh; None
    | 3 -> Document.dclose true dh; None
    | _ -> assert false (* property of dialogs *)
```

9.6 Save to file viewer

⟨Viewers.spec *other cases* 121a⟩+≡ (23h) ◁120a

```
  | Save      (* always save *)
```

⟨Viewers.reset() *setting other viewers* 121b⟩+≡ (112f) ◁118d

```
  Tkresource.stringlist "savedTypes" [] |> List.iter (fun ctype ->
    try
      let (typ,sub),_pars = Lexheaders.media_type ctype in
      Hashtbl.add viewers (typ,sub) Save
    with Http_headers.Invalid_header e ->
      Error.f (s_ "Invalid MIME type %s\n%s" ctype e)
  );
```

⟨Viewers.view *match viewer cases* 121c⟩+≡ (111b) ◁120b

```
  | Save ->
      Save.interactive (fun _ -> ()) dh;
      None
```

⟨*signature* Save.interactive 121d⟩≡ (335d)

```
  val interactive : (string -> unit) -> Document.handle -> unit
```

⟨*function* Save.interactive 121e⟩≡ (339b)

```
  let rec interactive cont dh =
    (* The initial content of the requester *)
    let url = Url.string_of dh.document_id.document_url in
    let path =
      match dh.document_id.document_url.path with Some p -> p | None -> "" in

    Fileselect.f (s_ "Save document") (function
      | [] ->
           (* by closing dh, we might break the cache *)
           dclose true dh
      | [fname] ->
          begin try
            let endmsg = (s_ "URL %s\nsaved as %s" url fname) in
            f cont dh fname endmsg;
            Document.add_log dh
                (s_ "Saving %s\nto %s" url fname)
                (* channel is not closed ! *)
                (fun () -> Msys.rm fname)
          with Sys_error msg ->
            Error.f (s_ "Cannot save to %s\n(%s)" fname msg);
            interactive cont dh
          end
      | _l -> raise (Failure "multiple selection")
      )
      "*"
      (Filename.basename path)
      false false
```

Chapter 10

HTML Display

⟨*signature* Htmlw.display_html 122a⟩≡ (387f)
```
  (* automatically added to list of viewers via toplevel stmt *)
  val viewer : Viewers.t
```

XXX

⟨Main.main() *misc initialisation* 122b⟩≡ (30c)
```
  (* Various stuff for the HTML viewer, needing Tk *)
  Ctext.init();
  Attrs.init !Textw_fo.html_bg; (* built the bullet images *)
```

10.1 The display info, Htmlw.display_html

⟨*class* Htmlw.display_html 122c⟩≡ (391b)
```
  class display_html ((top : Widget.widget),
                      (ctx : Viewers.context),
                      (_mediapars : (string * string) list),
                      (imgmanager: Imgload.loader),
                      (dh': Document.handle)) =
   object (self)
    inherit Viewers.display_info () as _di  (* gives us basic features *)
    inherit viewer_globs (ctx, dh')
    inherit html_parse (dh')
    inherit html_body () as _body
    inherit bored ()

    (* val imgmanager = imgmanager *)

    ⟨Htmlw.display_html private fields 123b⟩

    (*less: can be changed? *)
    val mutable title = Url.string_of ctx#base.document_url
    method di_title = title

    ⟨Htmlw.display_html frame widget methods 123a⟩
    ⟨Htmlw.display_html init method 124a⟩

    ⟨Htmlw.display_html load images methods 123f⟩
    ⟨Htmlw.display_html update embedded objects methods 164f⟩
    ⟨Htmlw.display_html load frames methods 158d⟩
    ⟨Htmlw.display_html fragment methods 199e⟩

    ⟨Htmlw.display_html error managment methods 160b⟩
    ⟨Htmlw.display_html progress method 205c⟩
    ⟨Htmlw.display_html abort methods 203f⟩
    ⟨Htmlw.display_html redisplay methods 204f⟩

    ⟨Htmlw.display_html graphic cache destroy methods 238c⟩
    ⟨Htmlw.display_html other methods or fields 159e⟩

  end
```

⟨Htmlw.display_html *frame widget methods* 123a⟩≡ (122c)

```
  val frame =
    if not (Winfo.exists top)
    then failwith "too late"
    else Frame.create_named top (Mstring.gensym "html") [Class "Html"]
      (* this might as well fail if the window was destroyed before we
       * finally could get the headers of the document.
       *)
  method frame = frame

  method di_widget = frame
```

⟨Htmlw.display_html *private fields* 123b⟩≡ (122c) 123d ▷

```
  val mutable init_mode = true
  val mutable pending = true
```

⟨Htmlw.display_html *reset private fields* 123c⟩≡ (124a) 123e ▷

```
  init_mode <- full;
  pending <- true;

  errors := [];
  annotations := [];
  terminated <- false;
  set_progress <- Progress.no_meter;
```

⟨Htmlw.display_html *private fields* 123d⟩+≡ (122c) ◁ 123b 159d ▷

```
  val mutable mach = F.create (ctx, imgmanager)
```

⟨Htmlw.display_html *reset private fields* 123e⟩+≡ (124a) ◁ 123c

```
  mach <- F.create (ctx, imgmanager);
```

⟨Htmlw.display_html *load images methods* 123f⟩≡ (122c) 171b ▷

```
  method mach = mach
```

⟨*module Htmlw.F* 123g⟩≡ (391b)

```
  module F = Html_disp (*Html_disp.Make(Textw_fo)(Form)(Table) *)
```

⟨*signature functor* Html_disp.Make 123h⟩≡ (423a)

```
  (*module Make
    (G: Htmlfmt.GfxHTML)
    (F: Htmlfmt.FormDisplay)
    (T: Htmlfmt.TableDisplay)
    : sig
      (* Do we need to export FormLogic and TableLogic so that extensions
       *  can access them ?
       *)
  *)
  val create : Viewers.context * imgloader -> machine
  (*end*)
```

```
⟨Htmlw.display_html init method 124a⟩≡                                  (122c)
  (* since we may be called multiple times, we have to clear some of the
     instance variables *)
  method init full =
    ⟨Htmlw.display_html reset private fields 123c⟩

    ⟨Htmlw.display_html set meta tag 135a⟩

    ⟨Htmlw.display_html set progress report and head UI 124c⟩

    self#body_init full;

    ⟨Htmlw.display_html display frames 158c⟩

    (* Asynchronous parsing and display, token by token *)
    self#parse_init;

    ⟨Htmlw.display_html i18 encoder for forms 226h⟩

    dh.document_feed.feed_schedule (fun () ->
      try
        let warnings, correct,   tokens, loc =
          lexer self#lexbuf
        in
        ⟨Htmlw.display_html in feed, record possible errors after lexing 160c⟩

        ⟨Htmlw.display_html in feed, annotate 159f⟩
         tokens |> List.iter (fun token ->
           begin
             try mach#send token
             with Html.Invalid_Html s -> self#record_error loc s
           end;
           if token = EOF then begin
             pending <- false;
             imgmanager#flush_images;
             raise End_of_file
           end
         )
      with
      | End_of_file ->
          ⟨Htmlw.display_html in feed, End_of_file exn handler 124b⟩
      ⟨Htmlw.display_html in feed, other exceptions handler 160d⟩
     )

⟨Htmlw.display_html in feed, End_of_file exn handler 124b⟩≡             (124a)
  self#set_progress (Some red) red;
  self#finish false;
  ⟨Htmlw.display_html in feed, End_of_file exn handler, goto fragment 199f⟩

⟨Htmlw.display_html set progress report and head UI 124c⟩≡             (124a)
  (* We have full display, so put up progress report and head UI *)
  if full then begin
    let hgbas, progf = progress_report frame ctx in
    set_progress <- progf;
    self#bored_init hgbas;
    pack [hgbas] [Side Side_Bottom; Fill Fill_X];

    let headgroup, set_title, add_link, add_header, add_ext_header =
      html_head_ui dh.dh_headers (fun () -> self#redisplay)
      self#current_scroll_mode frame ctx
    in
    add_extra_header <- add_ext_header;
    pack [headgroup] [Side Side_Top; Fill Fill_X];
    let set_title s =
      title <- s;
      set_title s
    in
    head_hook (headgroup, set_title, add_link, add_header) self#mach
  end;
```

10.2 The display machine, `Html_disp.machine`

⟨*functor* `Html_disp.Make` 125a⟩≡ (423b)

```
  (*module Make (G : GfxHTML) (F: FormDisplay) (T: TableDisplay) = struct
    module FormLogic = Html_form.Make(F)
    module TableLogic = Html_table.Make(T)
  *)
  module G = Textw_fo
  module FormLogic = Html_form
  module TableLogic = Html_table

    ⟨type Html_disp.Make.anchor_type 148a⟩

    (* Tag machinery *)
    ⟨type Html_disp.Make.html_behavior 126e⟩

    ⟨functions Html_disp.Make.ignore_xxx 134d⟩

    ⟨class Html_disp.Make.display_machine 125c⟩

    ⟨function Html_disp.Make.init 129a⟩

    ⟨function Html_disp.Make.create 125b⟩
  (* end *)
```

⟨*function* `Html_disp.Make.create` 125b⟩≡ (125a)

```
  let create (ctx, imgmanager) =
    let mach = new display_machine (ctx, imgmanager) in
    init mach;
    ⟨Html_disp.Make.create() run user hooks 188d⟩
    (mach :> machine)
```

10.2.1 class `Html_disp.display_machine`

⟨*class* `Html_disp.Make.display_machine` 125c⟩≡ (125a)

```
  class display_machine ((ctx : Viewers.context), imgmanager) =
   object (self)
      inherit machine ()

      (* Keep a copy of the arguments *)
      (* val ctx = ctx *)
      (* val imgmanager = imgmanager *)
      method ctx = ctx
      method imgmanager = imgmanager

      ⟨Html_disp.display_machine base methods 126b⟩
      ⟨Html_disp.display_machine target methods 126d⟩
      ⟨Html_disp.display_machine private fields 126f⟩

      ⟨Html_disp.display_machine html token input methods 128d⟩

      ⟨Html_disp.display_machine tag machinery methods 126h⟩
      ⟨Html_disp.display_machine action stack methods 127b⟩
      ⟨Html_disp.display_machine formatter methods 127d⟩

      ⟨Html_disp.display_machine embedded methods 163b⟩
      ⟨Html_disp.display_machine fragment methods 199h⟩

     ⟨Html_disp.display_machine i18n methods 227a⟩

  end
```

⟨Html_disp.machine *other fields* 126a⟩≡ (27b) 126c ▷

```
  method virtual base : string
  method virtual set_base : string -> unit
```

⟨Html_disp.display_machine *base methods* 126b⟩≡ (125c)

```
  val mutable base = Url.string_of ctx#base.document_url
  method base = base
  method set_base s = base <- s
```

⟨Html_disp.machine *other fields* 126c⟩+≡ (27b) ◁ 126a

```
  method virtual target : string option
  method virtual set_target : string -> unit
```

⟨Html_disp.display_machine *target methods* 126d⟩≡ (125c)

```
  val mutable target = None
  method target = target
  method set_target t = target <- Some t
```

⟨*type* Html_disp.Make.html_behavior 126e⟩≡ (125a)

```
  type html_behavior = {
    tag_open  : Htmlfmt.formatter -> Html.tag -> unit;
    tag_close : Htmlfmt.formatter -> unit
  }
```

⟨Html_disp.display_machine *private fields* 126f⟩≡ (125c) 126i ▷

```
  val (*private*) tags = (Hashtbl.create 101 : (string, html_behavior) Hashtbl.t)
```

⟨Html_disp.machine *tags methods* 126g⟩≡ (27b)

```
  method virtual get_tag :
    string ->
    (Htmlfmt.formatter -> Html.tag -> unit) *
    (Htmlfmt.formatter -> unit)
```

⟨Html_disp.display_machine *tag machinery methods* 126h⟩≡ (125c)

```
  (* Adding and removing tag behaviors *)
  method add_tag t o c =
    Hashtbl.add tags t {tag_open = o; tag_close = c}
  method get_tag t =
    let {tag_open = o; tag_close = c} = Hashtbl.find tags t in
    o,c
  method remove_tag = Hashtbl.remove tags
```

⟨Html_disp.display_machine *private fields* 126i⟩+≡ (125c) ◁ 126f 127c ▷

```
  val mutable (*private*) action = (fun _s -> ())
  val mutable (*private*) action_stack = []
```

⟨Html_disp.machine *action stack methods* 127a⟩≡ (27b)
```
  method virtual push_action : (string -> unit) -> unit
  method virtual pop_action : unit
```

⟨Html_disp.display_machine *action stack methods* 127b⟩≡ (125c)
```
  (* Changing the default mode for pcdata and cdata *)
  method push_action f =
    action_stack <-  f :: action_stack;
    action <- f
  method pop_action =
    match action_stack with
    | [] -> Log.f "Warning: empty action stack"
    | _old::l ->
    action_stack <- l;
    action <- match l with [] -> (fun _s ->()) | newa::_ -> newa
```

⟨Html_disp.display_machine *private fields* 127c⟩+≡ (125c) ◁126i 127e▷
```
  val mutable (*private*) formatter = default_fo
```

⟨Html_disp.display_machine *formatter methods* 127d⟩≡ (125c) 127g▷
```
  (* Accessing the variables *)
  method formatter = formatter
```

⟨Html_disp.display_machine *private fields* 127e⟩+≡ (125c) ◁127c 199i▷
```
  val mutable (*private*) formatter_stack = []
```

⟨Html_disp.machine *formatter stack methods* 127f⟩≡ (27b)
```
  method virtual push_formatter : Htmlfmt.formatter -> unit
  method virtual pop_formatter : Htmlfmt.formatter
```

⟨Html_disp.display_machine *formatter methods* 127g⟩+≡ (125c) ◁127d 127i▷
```
  (* Nested formatters for table cells and other usage *)
  method push_formatter fo =
    formatter <- fo;
    formatter_stack <- fo :: formatter_stack;
    self#push_action fo.format_string;
    see_frag <- fo.see_frag

  method pop_formatter =
    self#pop_action;
    match formatter_stack with
    | [] ->
        Log.f "Warning: empty formatter stack";
        default_fo
    | old::l ->
        old.flush();
        see_frag <- old.see_frag;
        formatter_stack <- l;
        formatter <- (match l with [] -> default_fo | newf :: _ -> newf);
        old
```

⟨Html_disp.machine *formatter misc methods* 127h⟩≡ (27b)
```
  method virtual create_formatter :
    Htmlfmt.formatterSpec -> Widget.widget -> Htmlfmt.formatter * Widget.widget
```

⟨Html_disp.display_machine *formatter methods* 127i⟩+≡ (125c) ◁127g 128a▷
```
  (* Nested windows *)
  val table_namer = Mstring.egensym "tablecell"
  method create_formatter spec w = G.create table_namer spec w ctx
```

⟨Html_disp.display_machine *formatter methods* 128a⟩+≡ (125c) ◁127i
```
  (* This is only for robustness *)
  method flush_formatters =
    while List.length formatter_stack > 0 do
      Log.f "WARNING: too many formatters in stack";
      self#pop_formatter.flush()
    done
```

⟨*function* Textfw_fo.create.flush 128b⟩≡ (129c)
```
  flush = (fun () ->
    fonts#pop_all (cur()); (* basefont lossage *)
    internal_flush true
  );
```

⟨Html_disp.machine *html input other methods* 128c⟩≡ (27b)
```
  method virtual look_for : Html.token -> unit
```

⟨Html_disp.display_machine *html token input methods* 128d⟩≡ (125c) 128e▷
```
  (* ignore everything up to some tag *)
  val mutable look_for = None

  method look_for e =
    look_for <- Some e
```

⟨Html_disp.display_machine *html token input methods* 128e⟩+≡ (125c) ◁128d
```
  (* Dispatching a token *)
  method private normal_send = function
  | Html.EOF -> self#flush_formatters;
  | CData s -> action s
  | PCData s -> action s
  | OpenTag t ->
      begin try
        let tag = Hashtbl.find tags t.tag_name in
        tag.tag_open formatter t
     with Not_found ->
       if !verbose
       then Logs.debug (fun m -> m "Display machine: <%s> ignored" t.tag_name)
     end
  | CloseTag n ->
      begin try
        (Hashtbl.find tags n).tag_close formatter
      with Not_found ->
         if !verbose
        then Logs.debug (fun m -> m "Display machine: </%s> ignored" n)
      end
  | Comment _ -> ()
  | Doctype _ -> ()

  method send tok =
    match look_for with
    | None -> self#normal_send tok
    | Some it when it = tok ->
        self#normal_send tok;
        look_for <- None
    | _ -> ()
```

10.2.2 Html_disp.Make.init()

```
⟨function Html_disp.Make.init 129a⟩≡                                    (125a)
  (* Standard initialisation for HTML 2.0 (+bits of 3.2) *)
  let init (mach : display_machine) =
    ⟨Html_disp.Make.init() HTML elements machine initialisation 134c⟩
  ()
```

10.3 The formatter

```
⟨signature Textw_fo.create 129b⟩≡                                      (422b)
  val create :
    (unit -> string) ->
    Htmlfmt.formatterSpec ->
    Widget.widget -> Viewers.context ->  Htmlfmt.formatter * Widget.widget
```

```
⟨function Textw_fo.create 129c⟩≡                                        (422g)
  (* Build a formatter, as required by html_disp *)
  let create namer spec top ctx =
    ⟨Textw_fo.create locals 130⟩
    let formatter =
    {
      ⟨function Textfw_fo.create.new_paragraph 138b⟩
      ⟨function Textfw_fo.create.close_paragraph 138c⟩
      ⟨function Textfw_fo.create.print_newline 151a⟩
      ⟨function Textfw_fo.create.print_verbatim 140d⟩
      ⟨function Textfw_fo.create.format_string 146b⟩

      ⟨function Textfw_fo.create.flush 128b⟩

      ⟨function Textfw_fo.create.hr 151e⟩
      ⟨function Textfw_fo.create.bullet 144c⟩


      ⟨function Textfw_fo.create.set_defaults 156⟩

      ⟨function Textfw_fo.create.push_attr 139d⟩
      ⟨function Textfw_fo.create.pop_attr 140a⟩

      ⟨function Textfw_fo.create.isindex 136c⟩

      ⟨function Textfw_fo.create.start_anchor 150c⟩
      ⟨function Textfw_fo.create.end_anchor 150d⟩

      ⟨function Textfw_fo.create.add_mark 149d⟩

      ⟨function Textfw_fo.create.create_embedded 170b⟩

      ⟨function Textfw_fo.create.see_frag 200a⟩
      } in

    formatter, fhtml
```

```
⟨type Htmlfmt.formatterSpec 129d⟩≡                                      (374c)
  type formatterSpec =
    | TopFormatter of bool  (* flag is pixel-scrolling mode *)
    | NestedFormatter
    | FrameFormatter of (string * string) list (* decoration ... *)
```

⟨Textw_fo.create *locals* 130⟩≡ (129c)

```
  let other_bg = ref (fun _ -> ()) in
  let fhtml, thtml =
    match spec with
      TopFormatter pscrolling ->
    let f,t =
     if pscrolling then begin
      let f,t =
        Ctext.create top [Wrap WrapWord; State Disabled] true in
        Canvas.configure (Winfo.parent t)
           [Background (NamedColor !html_bg)];
         other_bg := Canvas.configure (Winfo.parent t);
        f, t
      end
     else
       new_scrollable_text top
          [Wrap WrapWord; State Disabled]
          true
    in
    (* Try to solve focus problem -- JPF *)
    bind t [[],Enter] (BindSet ([], fun _ -> Focus.set t));
    f, t
    | NestedFormatter -> (* Embedded formatters (tables) *)
       let t = Text.create_named top (namer())
            [BorderWidth (Pixels 0); State Disabled;
              Relief Sunken; Wrap WrapNone;
          TextWidth 1; TextHeight 1]
    in
        t, t
    | FrameFormatter args ->
    let marginwidth =
      try [PadX (Pixels (int_of_string (List.assoc "marginwidth" args)))]
      with Not_found | Failure "int_of_string" -> []
    and marginheight =
      try [PadY (Pixels (int_of_string (List.assoc "marginheight" args)))]
      with Not_found | Failure "int_of_string" -> []
    in
   let f,t =
     Ctext.create top (marginwidth @ marginheight @
                [TextHeight 1;
                  Wrap WrapWord; State Disabled]) true in
   Canvas.configure (Winfo.parent t)
     [Background (NamedColor !html_bg)];
         other_bg := Canvas.configure (Winfo.parent t);

    f, t
  in

  (* Tk4.0pl3 fix, + avoid cb to scrollbar *)
  (* Make the widget searchable *)
  (* NOTE: search doesn't apply to nested windows *)
  begin match spec with
    TopFormatter _ | FrameFormatter _ ->
      Text.configure thtml [TakeFocus true; InsertOffTime 0];
      Frx_text.addsearch thtml
  | NestedFormatter ->
      Text.configure thtml [TakeFocus false; InsertOffTime 0]
  end;

  (* Set (other) defaults *)
```

```
let _, html_font = Fonts.compute_tag !Fonts.default in
 Text.configure thtml html_font;

(* transparent GIF hack *)
Textvariable.set (Textvariable.coerce "TRANSPARENT_GIF_COLOR") !html_bg;

(* The formatter
 *    to minimize calls to Tk, we write only one string for
 * each paragraph larger than some size. Because of this, it seems
 * that we also have to set tags and marks at the end.
 *)

(* Things queued *)
let marks = ref []
and embedded = ref []
and tagdefs = new Attrs.tags thtml
(* Hypertext Anchor support *)
and anchors = new Attrs.anchortags thtml

(* It's easier for us to keep character positions as offsets from the
 * beginning, but it's very costly in Tk (conversion 0+nchars -> index),
 * especially when the size gets large.
 * Thus, we keep the base index of the current buffer, and positions
 * as offsets from there.
 * We must be careful not to leave position values relative to old
 * buffer_base.
 *)
and buffer_base = ref (LineChar(0,0))
and position = ref 0
and anchor_start = ref (TextIndex(LineChar(0,0),[]))

(* Paragraphs and space squeezing *)
and trailing_space = ref false
and prev_is_newline = ref false
    (* if this is false, we are displaying text. if this is true, we
       just issued a newline *)
in

(* Index for Tk *)
let get_index p = TextIndex (!buffer_base, [CharOffset p]) in
let cur () = get_index !position in

(* colors for *this* window, can be changed by set_defaults *)
let fg = ref !html_fg
and bg = ref !html_bg
in
(* inherited properties (set_defaults) : we apply them to embedded
   formatters (table cells) *)
let inherited = ref [] in
anchors#init ctx;      (* install bindings *)
anchors#define "visited" [Foreground (NamedColor "MidnightBlue")];
anchors#define "anchor" [Foreground (NamedColor "#0000ff"); Underline true];


(* Size of buffer can impact performances *)
let refresh_threshold =
   if !internal_buffer < 1000 then 1000 else !internal_buffer in
let buffer = Ebuffer.create (2 * refresh_threshold)
and last_flush = ref !Low.global_time in
```

```
let internal_flush refresh = (* flush the buffer *)
  last_flush := !Low.global_time;
  Text.configure thtml [State Normal];
  Text.insert thtml textEnd (Ebuffer.get buffer) [];
  Ebuffer.reset buffer;
  List.iter
    (function (opts,p) -> Text.window_create thtml (get_index p) opts)
    (List.rev !embedded);
  List.iter (function (m,p) -> Text.mark_set thtml m (get_index p)) !marks;
  tagdefs#flush;
  anchors#flush;
  Text.configure thtml [State Disabled];
  marks := [];
  embedded := [];
   (* reset the position *)
  buffer_base :=
     Text.index thtml (TextIndex(!buffer_base,[CharOffset !position]));
  position := 0;
   (* try to give a reasonable initial height for the text widget *)
  if refresh then begin
   begin match spec with
     TopFormatter true
   | NestedFormatter
   | FrameFormatter _ -> Fit.set_initial_height thtml
   | _ -> ()
   end;
   Low.update_idletasks()
  end
   in

let put_text s =
  match String.length s with
    0 -> ()
  | l ->
      (* old: was Lexkanji.length but we needed to update the code to
       * handle Utf8 characters. Tk 8.x assumes utf8 characters so
       * if we compute here the position wrong, Tk would then highlight
       * anchors at the wrong position.
       *)
      position := !position + Mstring.utf8_length s;
      prev_is_newline := false;
      Ebuffer.output_string buffer s;
  trailing_space := s.[l-1] = ' ';
  if  !Low.global_time > !last_flush + 4 (* it's been a while *)
  then internal_flush true
  else if  Ebuffer.used buffer > refresh_threshold
  then internal_flush false
     in

(* Logic for tag manipulation *)
let margins = new Attrs.margin tagdefs
and aligns = new Attrs.align tagdefs
and fonts = new Attrs.font tagdefs
and fgcolors = new Attrs.fgcolor tagdefs
and bgcolors = new Attrs.bgcolor tagdefs
and spacing = new Attrs.spacing tagdefs
and offset = new Attrs.offset tagdefs
and underline = new Attrs.misc (tagdefs, "underline", [Underline true])
and strike = new Attrs.misc (tagdefs, "strike", [OverStrike true])
```

```
  in
  let put_embedded w align =
    let opts = match String.lowercase_ascii align with
      "top" -> [Align Align_Top]
    | "middle" -> [Align Align_Center] (* not exactly *)
    | "bottom" -> [Align Align_Baseline]
    |  _ -> [] in
    embedded := ((Window w)::opts, !position) :: !embedded;
    prev_is_newline := false;
    incr position    (* an embedded window is one char wide *)
  in

  let break () =
     if not !prev_is_newline then begin
       put_text "\n"; prev_is_newline := true
       end
  in

  let paropen = ref (cur()) in
```

10.4 HTML attributes

⟨*constant* Html.default_attributes 133a⟩≡ (295)

```
  (* Attribute values *)
  let default_attributes = [
    ("isindex"  , "prompt" ),  "Document is indexed/searchable: ";

    ("a"        , "methods"),  "GET";     (* <A METHODS=GET> *)
    ("embed"    , "methods"),  "GET";  (* <EMBED METHODS=GET> *)
    ("embed"    , "alt"    ),  "[EMBEDDED OBJECT]";(* <EMBED ALT="EMBEDDED OBJECT"> *)
    ("form"     , "method" ),  "GET";  (* <FORM METHOD=GET> *)
    ("form"     , "enctype"),  "application/x-www-form-urlencoded";

    ("ol"       , "type"   ),  "1";       (* <OL TYPE=1 *)
    ("input"    , "type"   ),  "TEXT"; (* <INPUT TYPE=TEXT> *)
    ("select"   , "size"   ),  "5";
    ("textarea" , "align"  ),  "bottom";
    ("input"    , "align"  ),  "bottom";
    ("select"   , "align"  ),  "bottom";
    ("img"      , "align"  ),  "bottom";
    (* ("img"   , "alt"    ),  "[IMAGE]"; *) (* Just "IMAGE" ? Boring... *)
    ("area"     , "shape"  ),  "rect";
    ("div"      , "align"  ),  "left";
    ("basefont" , "size"   ),  "3";

    (* frames *)
    ("frame"    , "frameborder" ), "0";
    ("frame"    , "scrolling"   ), "auto";
    ("frameset" , "rows"        ), "100%";
    ("frameset" , "cols"        ), "100%";
    ]
```

⟨*signature* Html.get_attribute 133b⟩≡ (293d)

```
  val get_attribute : tag -> string -> string
    (* [get_attribute tag attrib_name] *)
```

⟨*function* Html.get_attribute 133c⟩≡ (295)

```
  let get_attribute tag attr =
    try
      List.assoc attr tag.attributes
    with Not_found ->
      List.assoc (tag.tag_name, attr) default_attributes
```

⟨*signature* Html.has_attribute 134a⟩≡ (293d)

```
  val has_attribute : tag -> string -> bool
    (* [has_attribute tag attrib_name] *)
```

⟨*function* Html.has_attribute 134b⟩≡ (295)

```
  let has_attribute tag attr =
       List.mem_assoc attr tag.attributes
    || List.mem_assoc (tag.tag_name, attr) default_attributes
```

10.5 HTML tags

10.5.1 Header

`<html>`

⟨Html_disp.Make.init() *HTML elements machine initialisation* 134c⟩≡ (129a) 134e ▷

```
  (* 5.1 HTML *)
  mach#add_tag "html" ignore_open ignore_close;
```

⟨*functions* Html_disp.Make.ignore_xxx 134d⟩≡ (125a)

```
  let ignore_open = fun _ _ -> ()
  let ignore_close = fun _ -> ()
```

`<head>`

⟨Html_disp.Make.init() *HTML elements machine initialisation* 134e⟩+≡ (129a) ◁ 134c 134f ▷

```
  (*
   * 5.2 Head: <HEAD>
   * <!ENTITY % head.content "TITLE & ISINDEX? & BASE? %head.extra">
   * <!ENTITY % head.extra "& NEXTID?">
   * NOTE: this is now handled elsewhere
   *)
  mach#add_tag "head" ignore_open ignore_close;
```

`<title>`

⟨Html_disp.Make.init() *HTML elements machine initialisation* 134f⟩+≡ (129a) ◁ 134e 135b ▷

```
  (*
   * 5.2.1 Title: <TITLE>
   * assumes a unique Text token inside since
   * <!ELEMENT TITLE - -  (#PCDATA)*>
   * the title is not printed
   * NOTE: this is now handled elsewhere
   *)
  mach#add_tag "title" ignore_open ignore_close;
```

<meta>

⟨Htmlw.display_html *set meta tag* 135a⟩≡ (124a)
```
  let meta_charset = ref None in
  (* <META HTTP-EQUIV="Content-Type" CONTENT="*/*;CHARSET=*"> stuff *)
  if not !ignore_meta_charset then begin
    mach#add_tag "meta" (fun _fo tag ->
     try
       let h = Html.get_attribute tag "http-equiv" in
       let v = Html.get_attribute tag "content" in
       match String.lowercase_ascii h with
       | "content-type" ->
            begin try
              let (t,h), l = Lexheaders.media_type v in
              if String.lowercase_ascii t <> "text" ||
                 String.lowercase_ascii h <> "html" then begin
                   Log.f ("Unknown meta content-type = "^t^"/"^h);
                  raise Exit
              end;
              try
                List.iter (fun (h,v) ->
                  if String.lowercase_ascii h = "charset" then begin
                    let v = String.lowercase_ascii v in
                    Log.f ("MetaCharset detect : " ^ v);
                    begin try
                      let code =
                        let code = ref Charset.Unknown in
                        try List.iter (fun (x,c) ->
                           if Str.string_match (Str.regexp x) v 0 then begin
                             code := c;
                             raise Exit
                           end
                           ) Charset.encode_table ;
                           raise Not_found
                        with Exit -> !code
                      in
                      (* feed_read#set_code code; this does not work... *)
                      meta_charset := Some code
                    with Not_found ->
                      Log.f (v ^ ": I don't know this charset")
                    end;
                    raise Exit
                 end) l;
              with Exit -> ()
            with _ -> () (* if failed to parse, ignore it *)
          end
       | _ -> ()
     with Not_found -> ()
   )
   ignore_close
    end;
```

<base>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 135b⟩+≡ (129a) ◁134f 136a▷
```
  (*
   * 5.2.2 Base Address: <BASE>
   * TARGET is from PR-HTML4.0
   *)
  mach#add_tag "base"
```

```
    (fun _fo tag ->
      begin
        try mach#set_target (Html.get_attribute tag "target")
        with Not_found -> ()
      end;
      begin
        try mach#set_base (Html.get_attribute tag "href")
        with Not_found -> raise (Html.Invalid_Html "HREF required in BASE")
      end)
    ignore_close
  ;
```

<isindex>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 136a⟩+≡ (129a) ◁135b 136d▷

```
  (*
   * 5.2.3 Keyword Index: <ISINDEX>
   * HTML3.2: PROMPT attribute (default given)
   * NOTE: ISINDEX in HEAD is handled elsewhere, but we must keep it
   *       here because it may appear in BODY
   *)
  mach#add_tag "isindex"
     (fun fo t -> fo.isindex (Html.get_attribute t "prompt") mach#base)
     ignore_close
  ;
```

⟨Htmlfmt.formatter *structure primitives methods* 136b⟩≡ (26e) 149c▷

```
  isindex : string -> string -> unit;         (* <ISINDEX> *)
```

⟨*function* Textfw_fo.create.isindex 136c⟩≡ (129c)

```
  (* Compliance: text is not part of document ? *)
  isindex = (fun prompt base ->
    let f,e = Frx_entry.new_label_entry thtml prompt
        (function s ->
          ctx#goto { h_uri = "?" ^ Urlenc.encode s;
                     h_context = Some base;
                     h_method = GET;
                     h_params = []}
    ) in
    (* default size 0 ! *)
    Entry.configure e [TextWidth 20];
    put_embedded f "";
    put_text "\n"
  );
```

<link>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 136d⟩+≡ (129a) ◁136a 137a▷

```
  (*
   * 5.2.4 Link: <LINK>
   * 5.2.5 Associated Meta-information: <META>
   * 5.2.6 Next Id: <NEXTID>
   * NOTE: this is now handled elsewhere (only in HEAD)
   *)
  ["link"; "meta"; "nextid"] |> List.iter (fun t ->
    mach#add_tag t ignore_open ignore_close
  );
```

10.5.2 Content, <body>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 137a⟩+≡ (129a) ◁136d 137b▷
```
  (*
   * 5.3 Body: <BODY>
   * <!ENTITY % html.content "HEAD, BODY">
   * Note: with textw_fo, flush disables the text widget, so anything
     beyond </BODY> will not be displayed. Some documents have multiple
     bodies, or </BODY> before the end of the document. So we decide
     to completely ignore this tag. A stricter interpretation would be
     {tag_open = ...; tag_close = (fun fo -> fo.flush())};
     Our simpled minded minimization rules also introduce multiple BODY.
   *)
  mach#add_tag "body" ignore_open ignore_close;
```

10.5.3 Headings, <h1>, <h2>, etc

⟨Html_disp.Make.init() *HTML elements machine initialisation* 137b⟩+≡ (129a) ◁137a 138e▷
```
  (*
   * 5.4 Headings <H1> ... <H6>
   * <!ELEMENT ( %heading )  - -  (%text;)*>
   * Assume headings may contain typographic styles, anchors
   * HTML3.2
   * <!ATTLIST ( %heading )
   *         align  (left|center|right) #IMPLIED
   *         >
   *)

  ⟨Html_disp.Make.init() headings private variables 137c⟩
  ⟨function Html_disp.Make.init.open_header 137d⟩
  ⟨function Html_disp.Make.init.close_header 138d⟩
  [1;2;3;4;5;6] |> List.iter (fun headnum ->
    mach#add_tag (Printf.sprintf "h%d" headnum) (open_header headnum) close_header
  );
```

⟨Html_disp.Make.init() *headings private variables* 137c⟩≡ (137b)
```
  (* Private variables of header *)
  let header_size = ref 0 in
  let header_align = ref None in
```

⟨*function* Html_disp.Make.init.open_header 137d⟩≡ (137b)
```
  let open_header size =
   fun (fo : Htmlfmt.formatter) tag ->
    fo.new_paragraph() ;
    header_size := size;
    push_style fo (Printf.sprintf "header%d" size);
    try
      let align = Html.get_attribute tag "align" in
      fo.push_attr [Justification align];
      header_align := Some align
    with Not_found -> header_align := None
  in
```

⟨Htmlfmt.gattr *alignment cases* 137e⟩≡ (26c)
```
  | Justification of string
```

⟨Htmlfmt.formatter *primitives methods* 138a⟩≡ (26d) 140c ▷
```
  new_paragraph: unit -> unit;   (* Open a new paragraph *)
    (* make sure the following text will start on a new line *)
  close_paragraph: unit -> unit;   (* Close a paragraph *)
    (* make sure there is an eol after the current text *)
```

⟨*function* Textfw_fo.create.new_paragraph 138b⟩≡ (129c)
```
  new_paragraph = (fun () ->
    break();
    spacing#push (cur()) 5;
    paropen := cur()
  );
```

⟨*function* Textfw_fo.create.close_paragraph 138c⟩≡ (129c)
```
  close_paragraph = (fun () ->
    spacing#pop (cur()) 5;
    if (cur() = !paropen)
    then prev_is_newline := false;
    break()
  );
```

⟨*function* Html_disp.Make.init.close_header 138d⟩≡ (137b)
```
  let close_header fo =
    pop_style fo (Printf.sprintf "header%d" !header_size);
    fo.close_paragraph();
    match !header_align with
    | None -> ()
    | Some a -> fo.pop_attr [Justification a]
  in
```

10.5.4 Paragraphs, <p>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 138e⟩+≡ (129a) ◁ 137b 140b ▷
```
  (*
   * 5.5.1 Paragraph: <P>
   *   a bit approximative in HTML 2.0
   * HTML3.2
   * <!ATTLIST P
   *         align  (left|center|right) #IMPLIED
   *         >
   *)
  let paligns = ref [] in

  mach#add_tag "p"
    (fun fo tag ->
       fo.new_paragraph ();
       try
         let a = Html.get_attribute tag "align" in
         paligns := (Some a) :: !paligns;
         fo.push_attr [Justification a]
       with Not_found -> paligns := None :: !paligns)
    (fun fo ->
       fo.close_paragraph();
       match !paligns with
       | [] -> () (* that's an error actually *)
       | (Some a)::l ->
          fo.pop_attr [Justification a];
          paligns := l
       | None::l ->
```

```
         paligns := l)
  ;
```

10.5.5 Text styles part 1

⟨*function* Html_disp.push_style 139a⟩≡ (423b)

```
  (* Style abbreviation
   * TODO?: check stack.
   *)
  let push_style (fo : Htmlfmt.formatter) s =
    try fo.push_attr (Styles.get s)
    with Not_found -> Logs.warn (fun m -> m "Missing style : %s" s)
```

⟨*function* Html_disp.pop_style 139b⟩≡ (423b)

```
  let pop_style (fo : Htmlfmt.formatter) s =
    try fo.pop_attr (Styles.get s)
    with Not_found -> Logs.warn (fun m -> m "Missing style : %s" s)
```

⟨Htmlfmt.formatter *graphical attributes methods* 139c⟩≡ (26d) 155f▷

```
  push_attr : gattr list -> unit;
  pop_attr : gattr list -> unit;
```

⟨*function* Textfw_fo.create.push_attr 139d⟩≡ (129c)

```
  push_attr = (fun l ->
    let fis = ref [] in
    l |> List.iter (function
      | Font fi ->
          fis := fi :: !fis
      | Margin n ->
          margins#push (cur()) n
      | Justification a ->
          aligns#push (cur()) a
      | FgColor s ->
          if !usecolors
          then fgcolors#push (cur()) s
      | BgColor s ->
          if !usecolors
          then bgcolors#push (cur()) s
      | Spacing n ->
          spacing#push (cur()) n
      | Underlined ->
          underline#push (cur())
      | Striked ->
          strike#push (cur())
      | Superscript ->
          fis := (FontDelta (-2)) :: !fis;
          offset#push (cur()) 5
      | Lowerscript ->
          fis := (FontDelta (-2)) :: !fis;
          offset#push (cur()) (-5)
      );
    if !fis <> []
    then fonts#push (cur()) !fis;
  );
```

⟨*function* Textfw_fo.create.pop_attr 140a⟩≡ (129c)
```
  pop_attr = (fun l ->
    let fis = ref [] in
    l |> List.iter (function
      | Font fi -> fis := fi :: !fis
      | Margin n -> margins#pop (cur()) n
      | Justification a -> aligns#pop (cur()) a
      | FgColor s ->
          if !usecolors then fgcolors#pop (cur()) s
      | BgColor s ->
          if !usecolors then bgcolors#pop (cur()) s
      | Spacing n -> spacing#pop (cur()) n
      | Underlined -> underline#pop (cur())
      | Striked -> strike#pop (cur())
      | Superscript ->
                fis := (FontDelta (-2)) :: !fis;
                offset#pop (cur()) 5
      | Lowerscript ->
                fis := (FontDelta (-2)) :: !fis;
                offset#pop (cur()) (-5)
    );
    if !fis <> []
    then fonts#pop (cur()) !fis;
  );
```

<pre>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 140b⟩+≡ (129a) ◁138e 141a▷
```
  (*
   * 5.5.2 Preformatted Text : <PRE>
   *    TODO: optional attribute WIDTH
   *    should be fixed font, respecting newlines
   *    local styles authorized however (i.e. markup is parsed)
  *)
  (* 5.5.2.1 Example and Listing: <XMP>, <LISTING>
   *    deprecated anyway
   *)

  ["pre"; "listing"; "xmp"] |> List.iter (fun s ->
    mach#add_tag s
      (fun fo _tag ->
         fo.new_paragraph();
         push_style fo "verbatim";
         mach#push_action fo.print_verbatim)
      (fun fo ->
         pop_style fo "verbatim";
         fo.close_paragraph();
         mach#pop_action)
  );
```

⟨Htmlfmt.formatter *primitives methods* 140c⟩+≡ (26d) ◁138a 146a▷
```
  print_verbatim : string -> unit; (* Print as-is *)
```

⟨*function* Textfw_fo.create.print_verbatim 140d⟩≡ (129c)
```
  print_verbatim = (fun s ->
    put_text s;
    prev_is_newline := false
  );
```

`<address>`

⟨`Html_disp.Make.init()` *HTML elements machine initialisation* 141a⟩+≡ (129a) ◁140b 141b▷
```
  (*
   * 5.5.3 Address: <ADDRESS>
   *)
  mach#add_tag "address"
     (fun fo _tag -> fo.new_paragraph(); push_style fo "italic")
     (fun fo -> pop_style fo "italic"; fo.close_paragraph())
  ;
```

`<blockquote>`

⟨`Html_disp.Make.init()` *HTML elements machine initialisation* 141b⟩+≡ (129a) ◁141a 141d▷
```
  (*
   * 5.5.4 Block Quote: <BLOCKQUOTE>
   *)
  mach#add_tag "blockquote"
     (fun fo _tag ->
        fo.new_paragraph();
        push_style fo "italic";
        fo.push_attr [Margin 10])
     (fun fo ->
        pop_style fo "italic";
        fo.pop_attr [Margin 10];
        fo.close_paragraph())
  ;
```

⟨`Htmlfmt.gattr` *spacing cases* 141c⟩≡ (26c) 146c▷
```
  | Margin of int
```

10.5.6 Text styles part 2

⟨`Html_disp.Make.init()` *HTML elements machine initialisation* 141d⟩+≡ (129a) ◁141b 142a▷
```
  (*
   * 5.7.1.1 Citation: <CITE>
   * 5.7.1.2 Code: <CODE>
   * 5.7.1.3 Emphasis: <EM>
   * 5.7.1.4 Keyboard: <KBD>
   * 5.7.1.5 Sample: <SAMP>
   * 5.7.1.6 Strong Emphasis: <STRONG>
   * 5.7.1.7 Variable: <VAR>
   *)

  (* Different typographic styles, shared *)
  let italic_style t =
    mach#add_tag t
       (fun fo _tag -> push_style fo "italic")
       (fun fo -> pop_style fo "italic")
  in
  let fixed_style t =
    mach#add_tag t
      (fun fo _tag -> push_style fo "fixed")
      (fun fo -> pop_style fo "fixed")
  in
  let bold_style t =
    mach#add_tag t
      (fun fo _tag -> push_style fo "bold")
      (fun fo -> pop_style fo "bold")
  in
```

Main styles, <b>, <i>, <tt>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 142a⟩+≡ (129a) ◁141d 142b▷

```
  (*
   * 5.7.2.1 Bold: <B>
   * 5.7.2.2 Italic: <I>
   * 5.7.2.3 Teletype: <TT>
   *)
  bold_style "b";
  italic_style "i";
  fixed_style "tt";
```

More Italics, <em>, <cite>, <var>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 142b⟩+≡ (129a) ◁142a 142c▷

```
  ["cite"; "em"; "var"] |> List.iter italic_style;
```

More fixed, <code>, <kbd>, <samp>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 142c⟩+≡ (129a) ◁142b 142d▷

```
  ["code"; "kbd"; "samp"] |> List.iter fixed_style;
```

More bold, <strong>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 142d⟩+≡ (129a) ◁142c 142f▷

```
  bold_style "strong";
```

10.5.7 Text styles part 3

⟨Htmlfmt.gattr *style cases* 142e⟩≡ (26c) 142g▷

```
  | Underlined
```

Underline, <u>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 142f⟩+≡ (129a) ◁142d 142h▷

```
  (* Some HTML 3.2 flashy features *)
  mach#add_tag "u"
    (fun fo _t -> fo.push_attr [Underlined])
    (fun fo  -> fo.pop_attr [Underlined]);
```

Strike, <strike>

⟨Htmlfmt.gattr *style cases* 142g⟩+≡ (26c) ◁142e 143a▷

```
  | Striked
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 142h⟩+≡ (129a) ◁142f 143b▷

```
  mach#add_tag "strike"
    (fun fo _t -> fo.push_attr [Striked])
    (fun fo  -> fo.pop_attr [Striked]);
```

Superscript, <sup>

⟨Htmlfmt.gattr *style cases* 143a⟩+≡ (26c) ◁142g 143c▷
```
  | Superscript
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 143b⟩+≡ (129a) ◁142h 143d▷
```
  mach#add_tag "sup"
    (fun fo _t -> fo.push_attr [Superscript])
    (fun fo  -> fo.pop_attr [Superscript]);
```

Lowerscript, <sub>

⟨Htmlfmt.gattr *style cases* 143c⟩+≡ (26c) ◁143a
```
  | Lowerscript
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 143d⟩+≡ (129a) ◁143b 143e▷
```
  mach#add_tag "sub"
    (fun fo _t -> fo.push_attr [Lowerscript])
    (fun fo  -> fo.pop_attr [Lowerscript]);
```

10.5.8 Text styles part 4

Center, <center>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 143e⟩+≡ (129a) ◁143d 143f▷
```
  (* Some HTML 3.2 flashy features *)
  mach#add_tag "center"
      (fun fo _t -> fo.push_attr [Justification "center"])
      (fun fo -> fo.pop_attr [Justification "center"]);
```

Alignment, <div>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 143f⟩+≡ (129a) ◁143e 143g▷
```
  (* Some HTML 3.2 flashy features *)
    mach#add_tag "div"
        (fun fo t -> fo.push_attr [Justification (Html.get_attribute t "align")])
        (fun fo -> fo.pop_attr [Justification "whocares"]);
```

Lowerscript, <sub>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 143g⟩+≡ (129a) ◁143f 143h▷
```
  (* Some HTML 3.2 flashy features *)
    mach#add_tag "big"
        (fun fo _t -> fo.push_attr [Font (FontDelta 2)])
        (fun fo  -> fo.pop_attr [Font (FontDelta 2)]);
```

Lowerscript, <sub>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 143h⟩+≡ (129a) ◁143g 144a▷
```
  (* Some HTML 3.2 flashy features *)
    mach#add_tag "small"
        (fun fo _t -> fo.push_attr [Font (FontDelta (-2))])
        (fun fo  -> fo.pop_attr [Font (FontDelta (-2))]);
```

10.5.9 Lists

<ul>, <li>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 144a⟩+≡ (129a) ◁143h 145▷

```
  (*
   * 5.6.1 Unordered List: <UL>, <LI>
   * HTML3.2
   * <!ENTITY % ULStyle "disc|square|circle">
   *
   * <!ATTLIST UL -- unordered lists --
   *         type    (%ULStyle)   #IMPLIED   -- bullet style --
   *         compact (compact)    #IMPLIED   -- reduced interitem spacing --
   *         >
   *)
  let list_level = ref 0 in

  let open_list (fo : Htmlfmt.formatter) tag =
    fo.push_attr [Margin 10];
    incr list_level;
    let bullet =
      try Html.get_attribute tag "type"
      with Not_found ->
         (match !list_level mod 3 with
         | 1 -> "disc"
         | 2 -> "circle"
         | _ -> "square"
         )
    in
    let compact = Html.has_attribute tag "compact" in
    let first_line = ref true in
    fo.new_paragraph();
    mach#add_tag "li"
       (fun fo tag ->
          if !first_line
          then first_line := false
          else
            if compact
            then fo.print_newline false
            else fo.new_paragraph();
          let bullet = try Html.get_attribute tag "type" with Not_found -> bullet in
       fo.bullet bullet)
       (fun fo -> if not compact then fo.close_paragraph())
  in

  let close_list (fo : Htmlfmt.formatter) =
    decr list_level;
    fo.close_paragraph();
    fo.pop_attr [Margin 10];
    mach#remove_tag "li"
  in

  mach#add_tag "ul" open_list close_list;
```

⟨Htmlfmt.formatter *predefined images methods* 144b⟩≡ (26e) 151d▷

```
  bullet : string -> unit;
```

⟨*function* Textfw_fo.create.bullet 144c⟩≡ (129c)

```
  (* TODO *)
  bullet = begin
   let bulletsym = Mstring.egensym "bullet" in
   (fun s ->
     try
       let img = Hashtbl.find Attrs.bullet_table s in
       put_embedded (Label.create_named thtml (bulletsym())
                     [img; BorderWidth (Pixels 0);
                      Background (NamedColor !html_bg)]) ""
     with Not_found  -> put_text "*"
    )
  end;
```

<ol>, <li>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 145⟩+≡   (129a) ◁144a 147e▷

```
  (*
   * 5.6.2 Ordered List: <OL>, <LI>
   * HTML3.2
   * <!--
   *        Numbering style
   *     1   arabic numbers     1, 2, 3, ...
   *     a   lower alpha        a, b, c, ...
   *     A   upper alpha        A, B, C, ...
   *     i   lower roman        i, ii, iii, ...
   *     I   upper roman        I, II, III, ...
   *
   *     The style is applied to the sequence number which by default
   *     is reset to 1 for the first list item in an ordered list.
   * -->
   *
   * <!ENTITY % OLStyle "CDATA" -- "1|a|A|i|I" but SGML folds case -->
   *
   * <!ATTLIST OL -- ordered lists --
   *         type     (%OLStyle)  #IMPLIED   -- numbering style --
   *         start     NUMBER     #IMPLIED   -- starting sequence number --
   *         compact  (compact)   #IMPLIED   -- reduced interitem spacing --
   *         >
   *)

  let numbering_styles =
    ["1", string_of_int;
     "a", lowernumber;
     "A", uppernumber;
     "i", (function i -> String.lowercase_ascii (roman i));
     "I", roman
    ]
  in

  let nesting = ref [] in

  let open_nlist (fo : Htmlfmt.formatter) tag =
    let li_counter =
      ref (try int_of_string (Html.get_attribute tag "start")
           with _ -> 1)
    in
    fo.push_attr [Margin 10];
    nesting := li_counter :: !nesting;
    let thisnumbers = List.rev !nesting in
    let numbering =
       try List.assoc (Html.get_attribute tag "type") numbering_styles
       with Not_found -> string_of_int
    in
    let compact = Html.has_attribute tag "compact" in
    mach#add_tag "li"
      (fun fo tag ->
        fo.new_paragraph();
        if compact
        then fo.push_attr [Spacing 0];
        (* if value is given, use it as number *)
        begin
          try
            let n = int_of_string (Html.get_attribute tag "value") in
            match !nesting with
            | c::_ -> c := n
            | _ -> () (* assert false *)
          with Not_found | Failure "int_of_string" -> ()
        end;
        thisnumbers |> List.iter (fun i ->
          fo.format_string (numbering !i);
          fo.format_string "."
        ))
      (fun fo ->
        incr li_counter;
        if compact
        then fo.pop_attr [Spacing 0];
        fo.close_paragraph())
  in

  let close_nlist (fo : Htmlfmt.formatter) =
    fo.pop_attr [Margin 10];
    nesting :=
      (match !nesting with
      | [] -> []
      | _x::l -> l
      );
    mach#remove_tag "li"
  in

  mach#add_tag "ol" open_nlist close_nlist;
```

⟨Htmlfmt.formatter *primitives methods* 146a⟩+≡ (26d) ◁140c 150f▷

```
  format_string : string -> unit; (* Line wrap, newlines don't count *)
```

⟨*function* Textfw_fo.create.format_string 146b⟩≡ (129c)

```
  format_string = (fun s ->
    if not !prev_is_newline
    then (* we are in text *)
      put_text (Html.beautify !trailing_space s)
    else (* decide if we should start a text *)
      let bs = Html.beautify true s in
      if bs = ""
      then () (* it was all spaces *)
      else begin
        put_text bs;
        prev_is_newline := false
      end
  );
```

⟨Htmlfmt.gattr *spacing cases* 146c⟩+≡ (26c) ◁141c

```
  | Spacing of int
```

⟨*function* Html_disp.lowernumber 147a⟩≡ (423b)
```
  (* SMOP Utilities for OL numbering *)
  let lowernumber n =
    let rec f cur n =
      if n < 0 then cur
      else  f (String.make 1 (Char.chr (97 + n mod 26)) ^ cur) (n / 26 - 1)
    in
    if n <= 0 then "*" else f "" (n-1)
```

⟨*function* Html_disp.uppernumber 147b⟩≡ (423b)
```
  let uppernumber n =
    let rec f cur n =
      if n < 0 then cur
      else  f (String.make 1 (Char.chr (64 + n mod 26)) ^ cur) (n / 26 - 1)
    in
    if n <= 0 then "*" else f "" (n-1)
```

⟨*constant* Html_disp.romans 147c⟩≡ (423b)
```
  let romans = [|
    [| ""; "I"; "II"; "III"; "IV"; "V"; "VI"; "VII"; "VIII"; "IX" |];
    [| ""; "X"; "XX"; "XXX"; "XL"; "L"; "LX"; "LXX"; "LXXX"; "XC" |];
    [| ""; "C"; "CC"; "CCC"; "CD"; "D"; "DC"; "DCC"; "DCCC"; "CM" |];
    [| ""; "M"; "MM"; "MMM"; "*MMM"; "*MMM"; "*MMM"; "*MMM"; "*MMM"; "*MMM" |];
    |]
```

⟨*function* Html_disp.roman 147d⟩≡ (423b)
```
  let roman n =
    let rec r cur level n =
      if n = 0 then cur
      else if level > 3 then "*" ^ cur
      else r (romans.(level).(n mod 10) ^ cur) (succ level)  (n / 10)
    in if n <= 0 then "*" else r "" 0 n
```

<dir>, <menu>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 147e⟩+≡ (129a) ◁145 147f▷
```
  (*
   * 5.6.3 Directory List: <DIR>
   * 5.6.4 Menu List: <MENU>
   *  do as <UL>, but we should work on presentation
   *)
  mach#add_tag "dir" open_list close_list;
  mach#add_tag "menu" open_list close_list;
```

<dl>, <dt>, <dd>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 147f⟩+≡ (129a) ◁147e 148b▷
```
  (*
   * 5.6.5 Definition List: <DL>, <DT>, <DD>
   *)
  let open_dl (fo : Htmlfmt.formatter) tag =
    let compact = Html.has_attribute tag "compact" in
    fo.new_paragraph();
    fo.push_attr [Margin 10];

    if not compact then begin
      let prev_is_dt = ref false in

      mach#add_tag "dt"
```

```
        (fun fo _tag ->
           if not !prev_is_dt then begin
             fo.new_paragraph();
             prev_is_dt := true
           end else
             fo.print_newline false;
           push_style fo "bold")
        (fun fo -> pop_style fo "bold");

      mach#add_tag "dd"
        (fun fo _tag ->
           if !prev_is_dt then begin
             fo.close_paragraph();
             prev_is_dt := false
           end;
           fo.new_paragraph();
           fo.push_attr [Margin 20])
        (fun fo ->
            fo.pop_attr [Margin 20];
            fo.close_paragraph())

      end else begin
        (* if compact *)
        let first_item = ref true in

        mach#add_tag "dt"
          (fun fo _tag ->
             if not !first_item
             then fo.print_newline false
             else first_item := false;
             push_style fo "bold")
          (fun fo -> pop_style fo "bold");

        mach#add_tag "dd"
          (fun fo _tag ->
             if not !first_item
             then fo.print_newline false
             else first_item := false;
             fo.push_attr [Margin 20])
          (fun fo -> fo.pop_attr [Margin 20])
      end
  in

  let close_dl (fo : Htmlfmt.formatter) =
    fo.pop_attr [Margin 10];
    fo.close_paragraph();
    mach#remove_tag "dt";
    mach#remove_tag "dd";
  in

  mach#add_tag "dl" open_dl close_dl;
```

10.5.10 Anchors, <a>

⟨*type* Html_disp.Make.anchor_type 148a⟩≡ (125a)
```
  type anchor_type = HREF | NAME
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 148b⟩+≡ (129a) ◁147f 150e▷
```
  (*
   * 5.7.3 Anchor: <A>
   * Assumes anchors are not nested
   * Can be both HREF and NAME.
   *)

  let anchor_type = ref None in
  let anchor_link = ref (Hyper.default_link "") in
  let in_anchor = ref false in

  let open_anchor (fo : Htmlfmt.formatter) tag =
    anchor_type := None;
    ⟨Html_disp.Make.init.open_anchor() look for NAME attribute 149a⟩
    ⟨Html_disp.Make.init.open_anchor() look for HREF attribute 149e⟩
  in

  let close_anchor (fo : Htmlfmt.formatter) =
    match !anchor_type with
    ⟨Html_disp.Make.init.close_anchor() match anchor type cases 149b⟩
    | None -> raise (Html.Invalid_Html "Unmatched </A>")
  in
  mach#add_tag "a" open_anchor close_anchor;
```

Anchor, `<a name=...>`

⟨Html_disp.Make.init.open_anchor() *look for NAME attribute* 149a⟩≡ (148b)
```
  (* is there a NAME attribute ? *)
  begin
    try
      fo.add_mark (Html.get_attribute tag "name");
      anchor_type := Some NAME
    with Not_found -> ()
  end;
```

⟨Html_disp.Make.init.close_anchor() *match anchor type cases* 149b⟩≡ (148b) 150a ▷
```
  | Some NAME ->
      in_anchor := false;
      anchor_type := None
```

⟨Htmlfmt.formatter *structure primitives methods* 149c⟩+≡ (26e) ◁136b 150b ▷
```
  add_mark : string -> unit;
```

⟨*function* Textfw_fo.create.add_mark 149d⟩≡ (129c)
```
  (* WARNING: if anchor name is a standard tk name, such as end,
     we're f*cked, so we force # *)
  add_mark = (fun s -> marks := ("#"^s, !position) :: !marks );
```

Anchor, `<a href=...>`

⟨Html_disp.Make.init.open_anchor() *look for HREF attribute* 149e⟩≡ (148b)
```
  (* is there an HREF attribute ? (if both, anchor_type is set to HREF *)
  (* so that close_anchor does the right thing) *)
  begin
    try
      let href = Html.get_attribute tag "href" in
      let h_params =
        try ["target", Html.get_attribute tag "target"]
        with Not_found ->
          (match mach#target with
          | Some s -> ["target", s]
          | None -> []
          )
      in
      anchor_link := {
        h_uri = href;
        h_context = Some mach#base;
        h_method =
          (try Hyper.parse_method (Html.get_attribute tag "methods")
           with _ -> GET
          );
        h_params = h_params
      };
      in_anchor := true;
      anchor_type := Some HREF;
      fo.start_anchor ();
      (* push_style fo "anchor" *)
    with Not_found ->
      (match !anchor_type with
      | None -> raise (Html.Invalid_Html "Missing NAME or HREF in <A>")
      | _ -> ()
      )
  end
```

⟨Html_disp.Make.init.close_anchor() *match anchor type cases* 150a⟩+≡ (148b) ◁149b

```
  | Some HREF ->
      fo.end_anchor !anchor_link;
      (* pop_style fo "anchor"; *)
      in_anchor := false;
      anchor_type := None
```

⟨Htmlfmt.formatter *structure primitives methods* 150b⟩+≡ (26e) ◁149c

```
  start_anchor : unit -> unit;
  end_anchor : Hyper.link -> unit;
```

⟨*function* Textfw_fo.create.start_anchor 150c⟩≡ (129c)

```
  start_anchor = (fun () -> anchor_start := (cur()));
```

⟨*function* Textfw_fo.create.end_anchor 150d⟩≡ (129c)

```
  (* set the tag for the anchor *)
  end_anchor = (fun link -> anchors#add_anchor !anchor_start (cur()) link);
```

10.5.11 Breaks

Line breaks, <br>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 150e⟩+≡ (129a) ◁148b 151c▷

```
  (*
   * 5.8 Line break: <BR>
   *)
  mach#add_tag "br"
    (fun fo _tag -> fo.print_newline true)
    ignore_close
  ;
```

⟨Htmlfmt.formatter *primitives methods* 150f⟩+≡ (26d) ◁146a

```
  print_newline : bool -> unit;  (* Force a line break *)
```

⟨*function* Textfw_fo.create.print_newline 151a⟩≡ (129c)
```
  print_newline = (fun force ->
   if force then begin
     put_text "\n";
     trailing_space := true
   end
   else break()
  );
```

Horizontal rules, <hr>

⟨*type* Html.length 151b⟩≡ (295 293d)
```
  (* HTML length *)
  type length =
      Nolength
    | LengthPixels of int
    | LengthRatio of float
    | LengthRel of int
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 151c⟩+≡ (129a) ◁150e 152e▷
```
  (*
   * 5.9 Horizontal Rule: <HR>
   *)
  mach#add_tag "hr"
    (fun fo tag ->
       let width =
         try Html.length_of_string (Html.get_attribute tag "width")
         with Not_found -> Nolength
       in
       let height =
         try int_of_string (Html.get_attribute tag "size")
         with Not_found | Failure "int_of_string" -> 1
       in
       let solid = Html.has_attribute tag "noshade" in
       fo.print_newline false;
       fo.hr width height solid;
       fo.print_newline false)
    ignore_close
  ;
```

⟨Htmlfmt.formatter *predefined images methods* 151d⟩+≡ (26e) ◁144b
```
  hr : Html.length -> int -> bool -> unit;  (* [hr width height solid] *)
```

⟨*function* Textfw_fo.create.hr 151e⟩≡ (129c)
```
  hr = begin
    let hrsym = Mstring.egensym "hr" in
    (fun width height solid ->
      let fr = Hr.create_named thtml (hrsym()) width height solid in
      Frame.configure fr [Background (NamedColor !fg)];
      put_embedded fr ""
    )
  end;
```

⟨*signature* Html.length_of_string 151f⟩≡ (293d)
```
  val length_of_string : string -> length
```

⟨*function* Html.length_of_string 152a⟩≡ (295)
```
  (* Either size in pixels or ration in percent *)
  let length_of_string s =
    try
      let pos = String.index s '%' in
      try LengthRatio (float_of_string (String.sub s 0 pos) /. 100.)
      with Failure "int_of_string" -> Nolength
    with Not_found ->
      try
        let pos = String.index s '*' in
        if pos = 0
        then LengthRel 1
        else
          try LengthRel (int_of_string (String.sub s 0 pos))
          with Failure "int_of_string" -> Nolength
      with Not_found ->
        try LengthPixels (int_of_string s)
        with Failure "int_of_string" -> Nolength
```

10.5.12 Tables

⟨*type* Htmlfmt.width_constraint 152b⟩≡ (374c)
```
  (* Table manager *)
  type width_constraint =
    | TopWidth    (* toplevel window size*)
    | FixedWidth of int   (* width is given in pixels *)
    | UnknownWidth of (unit -> bool) (* constraint to satisfy *)
```

⟨*signature* Html_disp.attempt_tables 152c⟩≡ (423a)
```
  val attempt_tables : bool ref
```

⟨*constant* Html_disp.attempt_tables 152d⟩≡ (423b)
```
  (* Preference settings *)
  let attempt_tables = ref false
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 152e⟩+≡ (129a) ◁151c 152f▷
```
  (* TABLE support *)
  if !attempt_tables
  then TableLogic.init mach
  else begin
    let behave_as oldtag newtag =
       mach#add_tag newtag
         (fun _fo _t -> mach#send (OpenTag {tag_name = oldtag; attributes = []}))
         (fun _fo ->   mach#send (CloseTag oldtag))
    in
    (* use DL for tables *)
    behave_as "dl" "table";
    mach#add_tag "tr" ignore_open ignore_close;
    behave_as "dt" "th";
    behave_as "dd" "td"
  end;
```

10.5.13 Maps, `<map>`

⟨Html_disp.Make.init() *HTML elements machine initialisation* 152f⟩+≡ (129a) ◁152e 155e▷
```
  (* Some HTML 3.2 good features *)
  let areas = ref [] in
  let mapname = ref "" in
```

```
mach#add_tag "map"
  (fun _fo t ->
       (* the name of the map *)
       let absname =
         try
      let name = Html.get_attribute t "name" in
      (* we must get a normalized name here *)
        Hyper.string_of {h_uri = "#"^name; h_context = Some mach#base;
                h_method = GET; h_params = []}
         with
       Not_found ->
         Hyper.string_of (Hyper.default_link mach#base)
       in
       mapname := absname;
       areas := [];
       mach#add_tag "area"
      (fun _fo tag ->
         let shape = String.lowercase_ascii (Html.get_attribute tag "shape")
         and href =
       try Some (Html.get_attribute tag "href") with Not_found -> None
             and coords =
       try Maps.parse_coords (Html.get_attribute tag "coords")
       with _ -> []
             and alttxt =
               try Html.get_attribute tag "alt" with Not_found -> ""
             in
         let h_params =
       try ["target", Html.get_attribute tag "target"]
       with
         Not_found ->
           match mach#target with
             Some s -> ["target", s]
           | None -> []
         in
         match href with
         | None -> () (* this creates a HOLE. not yet supported *)
         | Some uri ->
          let link = Hyper.{h_uri = uri; h_context = Some mach#base;
                          h_method = GET; h_params = h_params} in
          let area =
          match shape with
           "default" -> Maps.{area_kind = Default; area_coords = [];
                             area_link = link; area_alt = alttxt}
         | "rect" -> Maps.{area_kind = Rect; area_coords = coords;
                          area_link = link; area_alt = alttxt}
         | "circle" -> Maps.{area_kind = Circle; area_coords = coords;
                            area_link = link; area_alt = alttxt}
         | "poly" -> Maps.{area_kind = Poly; area_coords = coords;
                          area_link = link; area_alt = alttxt}
         | _ -> Maps.{area_kind = Default; area_coords = [];
                     area_link = link; area_alt = alttxt} in
               areas := area :: !areas)
          ignore_close)

  (fun _fo ->
    mach#remove_tag "area";
    Maps.add !mapname !areas)
;
```

⟨*type* Maps.area_kind 154a⟩≡ (290a 289b)
```
  (* The active areas *)
  type area_kind =
   | Rect
   | Circle
   | Poly
   | Default
```

⟨*type* Maps.area 154b⟩≡ (290a 289b)
```
  (* The area *)
  type area = {
    area_kind : area_kind;
    area_coords : int list;
    area_link : Hyper.link;
    area_alt  : string
   }
```

⟨*type* Maps.map 154c⟩≡ (290a 289b)
```
  type map = area list
```

⟨*type* Maps.t 154d⟩≡ (290a 289b)
```
  (* We merge any kind of map, for we actually are going to support
     maps for arbitrary embedded objects
   *)
  type t =
    | ClientSide of Hyper.link  (* usemap link *)
    | ServerSide of Hyper.link  (* ismap *)
    | Direct of Hyper.link   (* inside an anchor *)
    | NoMap    (* no additionnal navigation *)
    | FormMap of (int * int -> Hyper.link)
```

⟨*type* Maps.map_status 154e⟩≡ (290a 289b)
```
  (* The table of client-side image maps *)
  type map_status =
   | KnownMap of map
   | RequestedMap of string
```

⟨*signature* Maps.parse_coords 154f⟩≡ (289b)
```
  val parse_coords : string -> int list
```

⟨*signature* Maps.get 154g⟩≡ (289b)
```
  val get : string -> map_status
```

⟨*signature* Maps.add 154h⟩≡ (289b)
```
  val add : string -> map -> unit
```

⟨*constant* Maps.table 154i⟩≡ (290a)
```
  let table = (Hashtbl.create 37 : (string, map_status) Hashtbl.t)
```

⟨*constant* Maps.coord_sep 154j⟩≡ (290a)
```
  (* Tolerance: official syntax is "," separated.
     We use instead "[ \t\n]+\|\([ \t\n]*,[ \t\n]*\)"
     that is non empty sequence of whitespace
          or comma with possible surrounding whitespace
   *)
  (* let coord_sep = Str.regexp "," *)
  let coord_sep = Str.regexp "[ \t\n]+\\|\\([ \t\n]*,[ \t\n]*\\)"
```

⟨*function* Maps.parse_coords 154k⟩≡ (290a)
```
  let parse_coords s =
    List.map int_of_string (Str.split coord_sep s)
```

⟨*function* Maps.add 155a⟩≡ (290a)
```
  let add name map =
    Log.debug (sprintf "Adding map : %s" name);
    try
      match Hashtbl.find table name with
        KnownMap _m -> Log.debug "Map already known !"
      | RequestedMap event ->
          Hashtbl.remove table name; (* remove it *)
          Hashtbl.add table name (KnownMap map); (* add its value *)
          !broadcast_backend event (* trigger all waiting people *)
    with
      Not_found -> (* nobody requested it *)
        Hashtbl.add table name (KnownMap map)
```

⟨*function* Maps.get 155b⟩≡ (290a)
```
  let get name =
    Log.debug (sprintf "Asking map : %s" name);
    try
      Hashtbl.find table name
    with
      Not_found ->
          let m = Mstring.gensym "map" in
            Hashtbl.add table name (RequestedMap m);
       RequestedMap m
```

10.5.14 Fonts

⟨Htmlfmt.gattr *font cases* 155c⟩≡ (26c)
```
  | Font of Fonts.fontInfo          (* mostly size and face *)
```

⟨Plain.plain#init *setup fonts* 155d⟩≡ (114c)
```
  (*
  (* pick up the fixed font *)
  let attrs_fixed   = Styles.get_font "fixed" in
  let attrs_default = Styles.get_font "default" in
  let fd =
    Fonts.merge (Fonts.merge !Fonts.default attrs_default) attrs_fixed in
  let (_, opts) = Fonts.compute_tag fd in
  Text.configure text opts;
  *)
```

<basefont>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 155e⟩+≡ (129a) ◁152f 157a▷
```
  mach#add_tag "basefont"
    (fun fo t ->
       try
         let n = int_of_string (Html.get_attribute t "size") in
         fo.set_defaults "font" [Font (FontIndex n)]
       with Not_found | Failure "int_of_string" ->
         raise (Html.Invalid_Html "invalide SIZE"))
    ignore_close;
```

⟨Htmlfmt.formatter *graphical attributes methods* 155f⟩+≡ (26d) ◁139c
```
  set_defaults : string -> gattr list -> unit;     (* bg, fg, links *)
```

⟨*function* Textfw_fo.create.set_defaults 156⟩≡ (129c)
```
  (* TODO : vlink *)
  set_defaults = (fun name attrs ->
    inherited := (name, attrs) :: !inherited;
    match name with
    | "background" ->
        attrs |> List.iter (function
        | BgColor s ->
            if !usecolors then
              let c = Attrs.html_color s in
              if Frx_color.check c then begin
                bg := c;
                Resource.add
                   (sprintf "Mmm%s*background" (Widget.name thtml))
                     c Interactive;
                Text.configure thtml [Background (NamedColor c)];
                !other_bg [Background (NamedColor c)]
              end
        | _ -> ())

    | "foreground" ->
        attrs |> List.iter (function
          | FgColor s ->
             if !usecolors then
               let c = Attrs.html_color s in
               if Frx_color.check c then begin
                 fg := c;
                 Resource.add
                   (sprintf "Mmm%s*foreground" (Widget.name thtml))
                     c Interactive;
                 Text.configure thtml [Foreground (NamedColor c)]
             end
          | _ -> ())

    | "link" ->
        attrs |> List.iter (function
          | FgColor s ->
              if !usecolors then
                let c = Attrs.html_color s in
                if Frx_color.check c then
                  anchors#change "anchor" [Foreground (NamedColor c)]
          | _ -> ())

    | "alink" ->
        attrs |> List.iter (function
          | FgColor s ->
              if !usecolors then
                let c = Attrs.html_color s in
                if Frx_color.check c then
                  anchors#change "visited" [Foreground (NamedColor c)]
          | _ -> ())

    | "font" ->
        attrs |> List.iter (function
          | Font (FontIndex x)  ->
              fonts#set_base (cur()) x
          | _ -> ())

    | _ -> ()
  );
```

<font>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 157a⟩+≡ (129a) ◁155e 157c▷
```
  let fontchanges = ref [] in

  mach#add_tag "font"
    (fun fo t ->
       let attrs = [] in
       let attrs =
         try
           let size = Html.get_attribute t "size" in
           let l = String.length size in
           if l = 0
           then raise Not_found
           else
             if size.[0] = '+'
             then
               (Htmlfmt.Font (FontDelta (int_of_string (String.sub size 1 (pred l)))))
                      :: attrs
             else
               if size.[0] = '-'
               then (Htmlfmt.Font (FontDelta (int_of_string size)))::attrs
               else (Htmlfmt.Font (FontIndex (int_of_string size)))::attrs
         with Not_found | Failure _ -> attrs
       in
       let attrs =
          try
            let color = Html.get_attribute t "color" in
            (Htmlfmt.FgColor color)::attrs
          with Not_found -> attrs
       in
       (* attrs may well be the empty list *)
       if attrs <> []
       then fo.push_attr attrs;
       fontchanges := attrs :: !fontchanges)

    (fun fo ->
       match !fontchanges with
       | [] -> raise (Html.Invalid_Html "unmatched </font>")
       | x::l ->
           fontchanges := l;
           if x <> []
           then fo.pop_attr x)
  ;
```

⟨Htmlfmt.gattr *color cases* 157b⟩≡ (26c) 374b▷
```
  | FgColor of string
```

10.5.15 <style>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 157c⟩+≡ (129a) ◁157a 158a▷
```
  (* Some HTML 3.2 obnoxious features *)
  (* STYLE, SCRIPT in HEAD: not managed here
     For some reason, script is also allowed in text by the DTD.
     Make sure we just dump the contents...
     We do the same for style, just in case people dont respect the DTD
   *)
  mach#add_tag "style"
    (fun _fo _t -> mach#push_action (fun _s -> ()))
    (fun _fo -> mach#pop_action);
```

10.5.16 <script>

⟨Html_disp.Make.init() *HTML elements machine initialisation* 158a⟩+≡ (129a) ◁157c 161b▷
```
  (* Some HTML 3.2 obnoxious features *)
  mach#add_tag "script"
    (fun _fo _t -> mach#push_action (fun _s -> ()))
    (fun _fo -> mach#pop_action);
```

10.5.17 Frames

⟨*constant* Htmlw.frames_as_links 158b⟩≡ (391b)
```
  (* Prefs globals *)
  let frames_as_links = ref false
```

⟨Htmlw.display_html *display frames* 158c⟩≡ (124a)
```
  if not !frames_as_links then
    Htframe.add_frames (self#load_frames) (fun () ->
      match body_frame with
      | None -> ()
      | Some f -> Tk.destroy f
    ) frame mach;
```

⟨Htmlw.display_html *load frames methods* 158d⟩≡ (122c)
```
  method load_frames frames =
    (* all targets defined in all framesets in this document *)
    let targets =
      frames |> List.map (fun ((frdesc : Htframe.frame), w) ->
            frdesc.frame_name, w)
    in
    frames |> List.iter (fun ((frdesc : Htframe.frame), w) ->
      let thistargets =
           ("_self", w) (* ourselves *)
        :: ("_parent", Winfo.parent w) (* our direct parent *)
        :: targets (* common targets *)
      in
      let ectx = ctx#for_embed frdesc.frame_params thistargets in
      (* add frame parameters and targets to our ctx *)
      (* NOTE: there is a redundancy between embed_frame and _self in ctx,
         but frames are only an instance of embedded objects so we should
         no rely on the existence of _self for embed display machinery *)
       mach#add_embedded {
         embed_hlink = {
           h_uri = frdesc.frame_src;
           h_context = Some (Url.string_of ctx#base.document_url);
           h_method = GET;
           h_params = []
         };
         embed_frame = w;
         embed_context = ectx;
         embed_map = Maps.NoMap;
         embed_alt = frdesc.frame_name
       }
    )
```

⟨*type* Viewers.frame_targets 158e⟩≡ (340c 339c)
```
  type frame_targets = (string * Widget.widget) list
```

⟨*signature* Viewers.frame_adopt 158f⟩≡ (339c)
```
  val frame_adopt : Widget.widget -> frame_targets -> frame_targets
      (* remap _self and _parent *)
```

⟨*signature* Viewers.frame_fugue 159a⟩≡ (339c)
```
  val frame_fugue : frame_targets -> frame_targets
      (* forget about _self and _parents *)
```

⟨*function* Viewers.frame_adopt 159b⟩≡ (340c)
```
  let frame_adopt w targets =
    targets |> List.map (function
      | "_self",_ -> "_self", w
      | "_parent", _ -> "_parent", Winfo.parent w
      | s, f -> s, f
    )
```

⟨*function* Viewers.frame_fugue 159c⟩≡ (340c)
```
  let frame_fugue targets =
    let rec ff accu = function
      | [] -> accu
      | ("_self", _) :: l -> ff accu l
      | ("_parent", _) :: l -> ff accu l
      | p :: l -> ff (p::accu) l
    in
    ff [] targets
```

10.6 XXX

10.6.1 Annotations

⟨Htmlw.display_html *private fields* 159d⟩+≡ (122c) ◁123d 159g▷
```
  val (*private*) annotations = ref []
```

⟨Htmlw.display_html *other methods or fields* 159e⟩≡ (122c) 159h▷
```
  method annotate loc = function
    | OpenTag {tag_name=name; _} ->
        annotations := (name, loc) :: !annotations
    | CloseTag name ->
        annotations := (name, loc) :: !annotations
    | _ -> ()
```

⟨Htmlw.display_html *in feed, annotate* 159f⟩≡ (124a)
```
  (* We annotate only the last token, which is normally the one
   * from the original token stream *)
  let rec annot_last = function
  | [] -> ()
  | [x] -> self#annotate loc x
  | _x::l -> annot_last l
  in
  annot_last tokens;
```

10.6.2 Extra headers

⟨Htmlw.display_html *private fields* 159g⟩+≡ (122c) ◁159d 160a▷
```
  val mutable add_extra_header = fun _f -> ()
```

⟨Htmlw.display_html *other methods or fields* 159h⟩+≡ (122c) ◁159e 192e▷
```
  method add_extra_header = add_extra_header
```

10.6.3 Error management

⟨Htmlw.display_html *private fields* 160a⟩+≡ (122c) ◁159g 203e▷
```
  val (*private*) errors = ref []
```

⟨Htmlw.display_html *error managment methods* 160b⟩≡ (122c)
```
  (* error reporting *)
  method record_error loc msg =
    errors := (loc,msg) :: !errors;
    self#set_progress size (-1)
```

⟨Htmlw.display_html *in feed, record possible errors after lexing* 160c⟩≡ (124a)
```
  warnings |> List.iter (fun (reason, pos) ->
    self#record_error (Html.Loc(pos,succ pos)) reason
  );
  (match correct with
  | Legal -> ()
  | Illegal reason -> self#record_error loc reason
  );
```

⟨Htmlw.display_html *in feed, other exceptions handler* 160d⟩≡ (124a) 160e▷
```
  | Html.Html_Lexing (s,n) ->
      (* this should not happen if Lexhtml was debugged *)
      self#record_error (Html.Loc(n,n+1)) s
```

⟨Htmlw.display_html *in feed, other exceptions handler* 160e⟩+≡ (124a) ◁160d
```
  | Unix.Unix_error(_,_,_) ->
      self#finish true
  | e ->
      Log.f (sprintf "FATAL ERROR (htmlw) %s" (Printexc.to_string e));
      self#finish true
```

Chapter 11

Embedded Elements

11.1 Overview

11.2 XXX

⟨*toplevel* Htmlw._2 161a⟩≡ (391b)
```
  let _ =
    Embed.add_viewer ("text", "html") embedded_viewer
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 161b⟩+≡ (129a) ◁158a 169f▷
```
  (* EMBED
   *  The definition is a mix of what was done for earlier versions
   *  of MMM and Netscape Navigator. The reason is to get compatible HTML for
   *  Caml Applets in both browsers.
   *)
  mach#add_tag "embed"
    (fun fo tag ->
       try
         let link = Hyper.{
           h_uri = Html.get_attribute tag "src";
           h_method = GET;
           h_context = Some mach#base;
           h_params = []} in
         let width =
          try Some (int_of_string (Html.get_attribute tag "width"))
          with Not_found -> None
         and height =
           try Some (int_of_string (Html.get_attribute tag "height"))
           with Not_found -> None
         and alttxt = Html.get_attribute tag "alt" in

         let fr = fo.create_embedded "" width height in
         mach#add_embedded {
         embed_hlink = link;
         embed_frame = fr;
         embed_context = mach#ctx#for_embed tag.attributes [];
         embed_map = NoMap; (* yet *)
         embed_alt = alttxt
        }
       with
        Not_found ->
         raise (Html.Invalid_Html ("SRC missing in EMBED")))
    ignore_close;
```

⟨*method* Viewers.context.for_embed 162a⟩≡ (24e)
```
  (* apply this on a copy ! *)
  method for_embed (vparams: vparams) (newtargets : frame_targets) : 'a =
    {< viewer_params = vparams;
       targets =
         let _oldtargets = targets in (* for debug *)
          match newtargets with
          (* keep exactly the same environment *)
          | [] -> targets
          (* assume I'm given new _self and _parent *)
          | l -> l @ frame_fugue targets
    >}
```

⟨*method* Viewers.context.in_embed 162b⟩≡ (24e)
```
  method in_embed did =
    {< base = did >}
```

⟨*nested function* Nav.stdctx.init.make_embed 162c⟩≡ (39)
```
  (* a new context for an embedded window *)
  let make_embed_ctx (w : Widget.widget) (targets : Viewers.frame_targets)
      : Viewers.context =
    let targets =
      ("_self", w) :: ("_parent", Winfo.parent w) :: (Viewers.frame_fugue targets) in
    let newctx = (new stdctx caps (did,nav))#init in
    begin
      try
        let f = List.assoc "pointsto" self#hyper_funs in
        let g = List.assoc "clearpointsto" self#hyper_funs in
        newctx#add_nav ("pointsto", f);
        newctx#add_nav ("clearpointsto", g);
      with Not_found -> ()
    end;
    (newctx#for_embed [] targets :> Viewers.context)
  in
```

⟨*nested function* Nav.stdctx.init.frame_goto 162d⟩≡ (39)
```
  let frame_goto ( caps : < Cap.network; .. > )
       (targets : Viewers.frame_targets) (hlink : Hyper.link) =
    try
    (* target semantics PR-HTML 4.0 16.3.2 *)
     match List.assoc "target" hlink.h_params with
     | "_blank" ->
      let w = Toplevel.create Widget.default_toplevel [] in
      Embed.add caps {
        embed_hlink = hlink;
        embed_frame = w;
        embed_context = make_embed_ctx w targets;
        embed_map = Maps.NoMap;
        embed_alt = "" }
     | "_self" ->
      let w = List.assoc "_self" targets in
      Embed.add caps {
        embed_hlink = hlink;
        embed_frame = w;
        embed_context = make_embed_ctx w targets;
        embed_map = Maps.NoMap;
        embed_alt = "" }
     | "_top" -> follow_link caps targets hlink
     | "_parent" ->
      let w = List.assoc "_parent" targets in
      Embed.add caps {
        embed_hlink = hlink;
        embed_frame = w;
        embed_context = make_embed_ctx w targets;
        embed_map = Maps.NoMap;
        embed_alt = "" }
     | s ->
      let w = List.assoc s targets in
      Embed.add caps {
        embed_hlink = hlink;
        embed_frame = w;
        embed_context = make_embed_ctx w targets;
        embed_map = Maps.NoMap;
        embed_alt = "" }
    with
     Not_found -> (* if we are in a frame, it is available as _self *)
    try
      let w = List.assoc "_self" targets in
      Embed.add caps {
        embed_hlink = hlink;
        embed_frame = w;
        embed_context = make_embed_ctx w targets;
        embed_map = Maps.NoMap;
        embed_alt = "" }
    with
      Not_found -> follow_link caps targets hlink
  in
```

⟨Html_disp.machine *embedded fields* 163a⟩≡ (27b)

```
  method virtual add_embedded : Embed.obj -> unit
  method virtual embedded : Embed.obj list
```

⟨Html_disp.display_machine *embedded methods* 163b⟩≡ (125c)

```
  (* record all embedded objects in this machine *)
  val mutable (*private*) embedded = []
  method add_embedded x =
    let caps = Cap.network_caps_UNSAFE () in
    Embed.add caps x;
    embedded <- x :: embedded
  method embedded = embedded
```

⟨Viewers.context *embedded methods signatures* 163c⟩≡ (24a)

```
  (*-*)
  method for_embed : vparams -> frame_targets -> 'a
  method in_embed : Document.id -> 'a
```

⟨Viewers.context *other methods signatures* 163d⟩≡ (24a)

```
  method params : vparams
```

⟨*type* Viewers.vparams 163e⟩≡ (340c 339c)

```
  (* list of additionnal parameters for the viewer, according to its
     activation point *)
  (* hyper functions are: "goto", "save", "gotonew" *)
  type vparams = (string * string) list
```

⟨*type* Embed.embobject 164a⟩≡ (333)
```
  (* Embedded objects *)
  type obj = {
    embed_hlink : Hyper.link; (* hyperlink to the object *)
    embed_frame : Widget.widget;
        (* the frame where the viewers can do their stuff *)
    embed_context : Viewers.context;
    embed_map : Maps.t; (* associated map *)
    embed_alt : string;
  }
```

⟨*signature* Embed.add 164b⟩≡ (333b)
```
  val add : < Cap.network; ..> ->
    obj -> unit
```

⟨*signature* Embed.update 164c⟩≡ (333b)
```
  val update : < Cap.network; ..> ->
    Widget.widget -> Viewers.context -> Document.t -> (unit -> unit) -> unit
```

⟨Viewers.display_info *embedded virtual methods signatures* 164d⟩≡ (25b)
```
  method virtual di_update : unit      (* update embedded objects *)
```

⟨Plain.plain *empty methods* 164e⟩≡ (113d) 169e ▷
```
  method di_update = ()
```

⟨Htmlw.display_html *update embedded objects methods* 164f⟩≡ (122c)
```
  method di_update =
    let caps = Cap.network_caps_UNSAFE () in
    imgmanager#update_images caps;

    Frx_after.idle (fun () ->
      mach#embedded |> List.iter (fun Embed.{embed_frame = f; _} ->
        Winfo.children f |> List.iter (Frx_synth.send "update")
      )
    )
```

11.3 Embedded viewers

⟨*type* Embed.viewer 164g⟩≡ (333)
```
  type viewer =
    Http_headers.media_parameter list ->
    Widget.widget -> Viewers.context -> Document.t -> unit
```

⟨*signature* Embed.add_viewer 164h⟩≡ (333b)
```
  val add_viewer :
    Http_headers.media_type -> viewer -> unit
```

⟨*signature* Embed.rem_viewer 164i⟩≡ (333b)
```
  val rem_viewer :  Http_headers.media_type -> unit
```

⟨*constant* Embed.embedded_viewers 164j⟩≡ (333c)
```
  let embedded_viewers = Hashtbl.create 11
```

⟨*function* Embed.embedded_viewer 165a⟩≡ (333c)
```
  let embedded_viewer (frame : Widget.widget) (ctx : Viewers.context)
      (doc : Document.t) : unit =
    (* Destroy the alt window *)
    List.iter Tk.destroy (Winfo.children frame);
    try
      let ctype = Http_headers.contenttype doc.document_headers in
      let (typ, subtyp), l = Lexheaders.media_type ctype in
      try
        let viewer =
          try Hashtbl.find embedded_viewers (typ, subtyp) with
          | Not_found -> Hashtbl.find embedded_viewers (typ, "*")
        in
        viewer l frame ctx doc
      with
      | Not_found ->
          let t = s_ "Embed Error: no viewer for type %s/%s" typ subtyp in
          let l = Label.create frame [ Text t ] in
          pack [ l ] []
    with
    | Not_found ->
        let t =
          s_ "Embed Error: no type for document %s"
            (Url.string_of doc.document_address)
        in
        let l = Label.create frame [ Text t ] in
        pack [ l ] []
    | Http_headers.Invalid_header e ->
        let t =
          s_ "Embed Error: malformed type %s (%s)"
            (Http_headers.contenttype doc.document_headers)
            e
        in
        let l = Label.create frame [ Text t ] in
        pack [ l ] []
```

⟨*constant* Embed.embedded 165b⟩≡ (333c)
```
  (* Remember all current embedded objects by their frame *)
  let embedded = (Hashtbl.create 101 : (string, obj) Hashtbl.t)
```

⟨*function* Embed.add_embed 165c⟩≡ (333c)
```
  (* add and notify *)
  let add_embed (emb : obj) =
    Hashtbl.add embedded (Widget.name emb.embed_frame) emb;
    Frx_synth.send "setembed" emb.embed_frame
```

⟨*function* Embed.when_destroyed 165d⟩≡ (333c)
```
  (* when the frame gets destroyed, remove us from the table *)
  let when_destroyed (w : Widget.widget) = Hashtbl.remove embedded (Widget.name w)
```

⟨*toplevel* Embed._1 165e⟩≡ (333c)
```
  let _ = Protocol.add_destroy_hook when_destroyed
```

⟨*function* Embed.add 165f⟩≡ (333c)
```
  (* Queueing an embed *)
  let add (caps : < Cap.network ; .. >) (emb : obj) =
    let {
      embed_hlink = link;
      embed_frame = frame;
      embed_context = embed_ctx;
      embed_map = _m;
      embed_alt = alt_txt;
    } =
      emb
    in
    (* Put up the ALT text *)
    List.iter Tk.destroy (Winfo.children frame);
    pack [ Label.create_named frame "alt" [ Text alt_txt ] ] [];
    (* Check if the type is defined and a viewer available *)
    try
      let given_type = List.assoc "type" embed_ctx#params in
      let (typ, subtyp), parms = Lexheaders.media_type given_type in
      try
        let viewer =
          try Hashtbl.find embedded_viewers (typ, subtyp) with
          | Not_found -> Hashtbl.find embedded_viewers (typ, "*")
        in
        EmbeddedScheduler.add_request
          (caps :> < Cap.network >)
          (Www.make link) embed_ctx#base
          (* the continuation: it will receive the document *)
            (fun _url doc ->
            let doc =
              Document.
                { doc with
                  document_headers =
                    Http_headers.merge_headers doc.document_headers
                      [ "Content-Type: " ^ given_type ];
                }
            in
            (* Destroy the alt window *)
            List.iter Tk.destroy (Winfo.children frame);
            (* Add to our table/notify *)
            add_embed emb;
            viewer parms frame embed_ctx doc)
          (Tk_progress.meter frame)
      with
      | Not_found ->
          (* no viewer for this *)
          let t = s_ "Embed Error: no viewer for type %s" given_type in
          pack [ Label.create frame [ Text t ] ] []
      | Www.Invalid_request (w, msg) ->
          let t = s_ "Embed Error: %s\n(%s)" (Url.string_of w.www_url) msg in
          pack [ Message.create frame [ Text t ] ] []
      | Hyper.Invalid_link _err ->
          let t = s_ "Embed Error: invalid link" in
          pack [ Message.create frame [ Text t ] ] []
    with
    | Not_found -> (
        (* not type given, we have to retrieve to know *)
        (* Firing the request *)
        try
          EmbeddedScheduler.add_request
            (caps :> < Cap.network >)
            (Www.make link) embed_ctx#base
            (* the continuation: it will receive the document *)
            (* In general, we don't know the type before we get the document *)
              (fun _url doc -> embedded_viewer frame embed_ctx doc)
            (Tk_progress.meter frame)
          with
          | Www.Invalid_request (w, msg) ->
              let t = s_ "Embed Error: %s\n(%s)" (Url.string_of w.www_url) msg in
              pack [ Message.create frame [ Text t ] ] []
          | Hyper.Invalid_link _err ->
              let t = s_ "Embed Error: invalid link" in
              pack [ Message.create frame [ Text t ] ] [])
```

⟨*function* Embed.update 167⟩≡ (333c)

```
  let update (caps : < Cap.network ; .. >) (frame : Widget.widget)
      (embed_ctx : Viewers.context) (doc : Document.t) notchanged =
    try
      (* find the date of previous download, (or last-modified ?) *)
      let date_received = Http_headers.get_header "date" doc.document_headers in
      let rewrite_wr (wr : Www.request) =
        wr.www_headers <-
          ("If-Modified-Since: " ^ date_received) :: wr.www_headers;
        wr.www_headers <- "Pragma: no-cache" :: wr.www_headers;
        wr
      in
      let link = Hyper.default_link (Url.string_of doc.document_address) in
      (* wrapped viewer : decide if we need to redisplay or not *)
      let smart_viewer stdviewer frame embed_ctx (newdoc : Document.t) =
        let newdate =
          try Http_headers.get_header "date" newdoc.document_headers with
          | Not_found -> "foo"
        in
        if newdate <> date_received then begin
          List.iter Tk.destroy (Winfo.children frame);
          stdviewer frame embed_ctx newdoc
        end
        else notchanged ()
      in
      (* Check if the type is defined and a viewer available *)
      try
        let given_type = List.assoc "type" embed_ctx#params in
        let (typ, subtyp), parms = Lexheaders.media_type given_type in
        try
          let viewer =
            try Hashtbl.find embedded_viewers (typ, subtyp) with
            | Not_found -> Hashtbl.find embedded_viewers (typ, "*")
          in
          EmbeddedScheduler.add_request
            (caps :> < Cap.network >)
            (rewrite_wr (Www.make link))
            embed_ctx#base
            (* the continuation: it will receive the document *)
              (fun _url doc ->
              let doc =
                Document.
                  {
                    document_address = doc.document_address;
                    document_data = doc.document_data;
                    document_headers =
                      Http_headers.merge_headers doc.document_headers
                        [ "Content-Type: " ^ given_type ];
                  }
              in
              smart_viewer (viewer parms) frame embed_ctx doc)
            (Tk_progress.meter frame)
    with
    | Not_found ->
        (* no viewer for this *)
        let t = s_ "Embed Error: no viewer for type %s" given_type in
        pack [ Label.create frame [ Text t ] ] []
    | Www.Invalid_request (w, msg) ->
        let t = s_ "Embed Error: %s\n(%s)" (Url.string_of w.www_url) msg in
        pack [ Message.create frame [ Text t ] ] []
    | Hyper.Invalid_link _err ->
        let t = s_ "Embed Error: invalid link" in
        pack [ Message.create frame [ Text t ] ] []
  with
  | Not_found -> (
      (* not type given, we have to retrieve to know *)
      (* Firing the request *)
      try
        EmbeddedScheduler.add_request
          (caps :> < Cap.network >)
          (rewrite_wr (Www.make link))
          embed_ctx#base
          (* the continuation: it will receive the document *)
          (* In general, we don't know the type before we get the document *)
            (fun _url doc -> smart_viewer embedded_viewer frame embed_ctx doc)
          (Tk_progress.meter frame)
      with
      | Www.Invalid_request (w, msg) ->
          let t = s_ "Embed Error: %s\n(%s)" (Url.string_of w.www_url) msg in
          pack [ Message.create frame [ Text t ] ] []
      | Hyper.Invalid_link _err ->
          let t = s_ "Embed Error: invalid link" in
          pack [ Message.create frame [ Text t ] ] [])
with
| Not_found ->
    (* Document has no Date: header *)
    notchanged ()
```

Chapter 12

Images

⟨Html_disp.machine *image methods* 169a⟩≡ (27b)
```
  method virtual imgmanager : imgloader
```

⟨*Document menu elements* 169b⟩≡ (46d) 192a ▷
```
  [Label (s_ "Load Images")    ; Command load_images];
```

⟨*function* Mmm.navigator.load_images 169c⟩≡ (35a)
```
  let load_images () =
    match !current_di with
    | None -> ()
    | Some di -> di#di_load_images
  in
```

⟨Viewers.display_info *images virtual methods signatures* 169d⟩≡ (25b)
```
  method virtual di_load_images : unit (* load images *)
```

⟨Plain.plain *empty methods* 169e⟩+≡ (113d) ◁ 164e 199d ▷
```
  method di_load_images = ()
```

⟨Html_disp.Make.init() *HTML elements machine initialisation* 169f⟩+≡ (129a) ◁ 161b 173b ▷
```
  (*
   * 5.10 Image: <IMG>
   *)
  mach#add_tag "img"
    (fun fo tag ->
       try
         let src = Html.get_attribute tag "src" in
         let align = Html.get_attribute tag "align" in

         let width =
           try Some (int_of_string (Html.get_attribute tag "width"))
           with Not_found | Failure "int_of_string" -> None
         in
         let height =
           try Some (int_of_string (Html.get_attribute tag "height"))
           with Not_found | Failure "int_of_string" -> None
         in

         ⟨Html_disp.Make.init() IMG case, let alt 171a⟩
         let w = fo.create_embedded align width height in
         let link = Hyper.{
           h_uri = src;
           h_context = Some mach#base;
           h_method = GET;
           h_params = []
```

```
          } in
          ⟨Html_disp.Make.init() IMG case, let map 170c⟩
          let caps = Cap.network_caps_UNSAFE () in
          mach#imgmanager#add_image caps
            { embed_hlink = link;
              embed_frame = w;
              embed_context = mach#ctx#for_embed tag.attributes [];
              embed_map = map;
              embed_alt = alt
             }
        with Not_found -> (* only on SRC *)
          raise (Html.Invalid_Html "missing SRC in IMG"))

      ignore_close
    ;
```

⟨Htmlfmt.formatter *embedding primitives methods* 170a⟩≡ (26e)

```
  create_embedded :
     string -> int option -> int option -> Widget.widget;
       (* [create_embedded align w h ]:
      returns a widget that we can pass as target to the embed manager.
      Should respect background color ?
    *)
```

⟨*function* Textfw_fo.create.create_embedded 170b⟩≡ (129c)

```
  create_embedded = begin
    (* avoid space leak in Tk hash table : reuse the same names *)
    let embsym = Mstring.egensym "emb" in

    (fun _a w h ->
       let f = Frame.create_named thtml (embsym()) [Class "HtmlEmbedded"] in
       if !usecolors
       then Frame.configure f [Background (NamedColor !bg)];

       (* To solve the focus problem
       Tk.bindtags  f ((WidgetBindings thtml) :: Tk.bindtags_get f);
       bind f [[],Enter] (BindSet ([], fun _ -> Focus.set thtml));
        *)
       (* -- end *)
       (match w, h with
        | Some w, Some h ->
            Frame.configure f [Width (Pixels w); Height (Pixels h);
            BorderWidth (Pixels 0)];
             Pack.propagate_set f false
        | _, _ -> ()
       );
       put_embedded f "";
       f
    )
  end;
```

⟨Html_disp.Make.init() *IMG case, let map* 170c⟩≡ (169f)

```
  (* some people use both ismap and usemap...
   *  prefer usemap
   *)
  let map =
    try
      let mapname = Html.get_attribute tag "usemap"  in
      Maps.ClientSide {
        h_uri = mapname;
        h_context = Some mach#base;
        h_method = GET;
        h_params = []
      }
    with Not_found ->
      if !in_anchor
      then
        if Html.has_attribute tag "ismap"
        then Maps.ServerSide !anchor_link
        else Maps.Direct !anchor_link
      else NoMap
  in
```

⟨Html_disp.Make.init() *IMG case, let alt* 171a⟩≡ (169f)

```
  let alt =
    try Html.get_attribute tag "alt"
    with Not_found ->
      let image_name =
        let pos =
          let cpos = ref (String.length src) in
          try
            while !cpos > 0 do
              match src.[!cpos - 1] with
              | '/' | '\\' (* for f!@#ing DOS users *) -> raise Exit
              | _ -> decr cpos
            done;
            0
         with Exit -> !cpos
       in
       if pos = String.length src
       then "IMAGE"
       else String.sub src pos (String.length src - pos)
    in
    Printf.sprintf "[%s]" image_name
  in
```

⟨Htmlw.display_html *load images methods* 171b⟩+≡ (122c) ◁123f

```
  method di_load_images =
    (* load our images *)
    imgmanager#load_images;

    (* Recursively, for all embedded objects, send the load_images event.
     * Because we work on the children of the frame, the *currently*
     *  displayed document in this frame gets the event.
     * NOTE: because of textvariable handlings, we can NOT send the
     *  event again during the processing of the event...
     *)
    Frx_after.idle (fun () ->
      mach#embedded |> List.iter (fun Embed.{ embed_frame = f; _} ->
        Winfo.children f |> List.iter (Frx_synth.send "load_images")
      )
    )
```

12.1 The image loader

⟨*signature class* Html_disp.imgloader 171c⟩≡ (423a)

```
  class  virtual imgloader : (unit) -> object
    ⟨Html_disp.imgloader virtual fields signatures 172a⟩
  end
```

⟨Html_disp.imgloader *virtual fields signatures* 172a⟩≡ (172b 171c)
```
  method virtual add_image : < Cap.network > -> Embed.obj -> unit
  method virtual flush_images : unit          (* flush when document is loaded *)
  method virtual load_images : unit   (* manual flush *)
  method virtual update_images : < Cap.network > -> unit
```

⟨*class* Html_disp.imgloader 172b⟩≡ (423b)
```
  class  virtual imgloader (_unit : unit) =
   object
    ⟨Html_disp.imgloader virtual fields signatures 172a⟩
  end
```

⟨*signature* Imgload.create 172c⟩≡ (376c)
```
  (* depending on !mode and !no_images will return a different kind of loader *)
  val create : unit -> loader
```

⟨*class* Imgload.loader *signature* 172d⟩≡ (376c)
```
  class loader : (unit) -> object
    method add_image : <Cap.network > ->
       Embed.obj -> unit       (* add one image *)

    method flush_images : unit          (* flush when document is loaded *)
    method load_images : unit  (* manual flush *)

    method update_images : < Cap.network> -> unit
  end
```

12.2 convert fallback

Chapter 13

Forms

⟨*type* `Htmlfmt.input_kind` 173a⟩≡ (374c)

```
(* Form manager *)
type input_kind = EntryInput | FileInput | OtherInput
```

⟨`Html_disp.Make.init()` *HTML elements machine initialisation* 173b⟩+≡ (129a) ◁169f

```
(* FORMS: they are defined elsewhere (html_form) *)
FormLogic.init mach;
(* standard basic HTML2.0 initialisation stops here *)
```

Chapter 14

CSS

14.1 Lexing

14.2 Parsing

14.3 Evaluating

Chapter 15

Javascript

15.1 Lexing

15.2 Parsing

15.3 Evaluating

Chapter 16

Applets

⟨*signature* Appview.code_viewer 176a⟩≡ (484a)
```
  val viewer: Viewers.t
```

⟨*signature* Appview.applet_viewer 176b⟩≡ (484a)
```
  val embed_viewer: Embed.viewer
```

16.1 hello_applet.cmo

⟨Http_headers.suffixes *elements* 176c⟩≡ (116c) 217a▷
```
  "cmo", ContentType "Content-Type: application/x-caml-applet; encoding=bytecode";
```

16.2 Data structures

⟨*type* Dload.t 176d⟩≡ (494 493)
```
  (* The "foreign" module fake cache : what we keep in memory *)
  type t = {
    module_address : string;  (* the URL of the bytecode *)
    module_info : string list;  (* headers *)
    module_functions : (string, applet_callback) Hashtbl.t
    }
```

⟨*type* Dload.applet_callback 176e⟩≡ (494 493)
```
  (* The type of entry point functions registered by the applet *)
  type applet_callback = Widget.widget -> Viewers.context -> unit
```

⟨*type* Applets.applet_callback 176f⟩≡ (483a 482)
```
  type applet_callback = Widget.widget -> Viewers.context -> unit
```

16.3 Initialisation

⟨Main.main() *applet system initialisation* 176g⟩≡ (30c)
```
  (* The applet system.
   * This loads the local modules also, so any setup that might be
   * overriden by a local module should happen before here.
   * However, preference initialisation must happen *after* initialisation
   * of the applet system
   *)
  Appsys.init !modules;
```

⟨*signature* Appsys.init 177a⟩≡ (483c)
```
  val init: bool (* whether to load also initial modules *) -> unit
```

⟨*function* Appsys.init 177b⟩≡ (483b)
```
  (* Dynamic linking init : both common applets and specific applets *)
  let init initialp =
    Logs.info (fun m -> m "Loading applet system");
    ⟨Appsys.init() sandbox restrictions 186b⟩
    ⟨Appsys.init() load local applets 188i⟩
    ⟨Appsys.init() plug mmm applets preferences 177c⟩
```

⟨Appsys.init() *plug mmm applets preferences* 177c⟩≡ (177b)
```
  Mmmprefs.plug_applets applets_pref
```

⟨*function* Appsys.applets_pref 177d⟩≡ (483b)
```
  (* Preference panel for applets *)
  let applets_pref top =
    Prefs.family top (I18n.sprintf "Applets") [
      Prefs.abstract_bool_pref "Active" pref_init pref_set;
        ⟨Appsys.applets_pref other elements 187e⟩
      ]
```

⟨*function* Appsys.pref_init 177e⟩≡ (483b)
```
  (* set display from prefs *)
  let pref_init v =
    Textvariable.set v (if !active then "1" else "0")
```

⟨*constant* Appsys.active 177f⟩≡ (483b)
```
  (* Pref stuff *)
  let active = ref false
```

⟨*function* Appsys.pref_set 177g⟩≡ (483b)
```
  let pref_set v =
    match Textvariable.get v with
    | "1" -> if !active then () else begin
               active := true;
               activate()
              end
    | _ -> if not !active then () else begin
             active := false;
             deactivate()
           end
```

⟨*constant* Appsys.types 177h⟩≡ (483b)
```
  let types = [
    "application","x-caml-applet"
  ]
```

⟨*function* Appsys.activate 177i⟩≡ (483b)
```
  let activate () =
    types |> List.iter (fun ctype ->
      Viewers.add_viewer ctype Appview.viewer;
      Embed.add_viewer ctype Appview.embed_viewer
    )
```

⟨*function* Appsys.deactivate 177j⟩≡ (483b)
```
  let deactivate () =
    types |> List.iter (fun ctype ->
      Viewers.rem_viewer ctype;
      Embed.rem_viewer ctype
    )
```

16.4 Viewers

16.4.1 Regular viewer

⟨*function* Appview.code_viewer 178a⟩≡ (483d)
```
(* If we load code directly (e.g. by clicking on a link pointing to a
   bytecode file).
   If the bytecode has been alread loaded, don't do anything.
   Else, proceed to load it (no entry point available !).
 *)
let viewer : Viewers.t =
  fun _parms frame (ctx : Viewers.context) (dh : Document.handle)
     : Viewers.display_info option ->
  let url = dh.document_id.document_url in
  try
    match Dload.get url with
    ⟨Appview.viewer() match Dload.get cases 178d⟩
  with
    ⟨Appview.viewer() exn handler 181e⟩
```

⟨*signature* Dload.get 178b⟩≡ (494)
```
val get : Url.t -> mod_status
```

⟨*type* Dload.mod_status 178c⟩≡ (494 493)
```
type mod_status =
  | Loaded of t
  | Unavailable of string list
  | Rejected of string list
```

⟨Appview.viewer() *match* Dload.get *cases* 178d⟩≡ (178a) 179d ▷
```
| Loaded cmo ->
  ⟨Appview.viewer() in Loaded case, if is_update 179a⟩
  else
  begin
    Document.dclose true dh;
    let f = Frame.create frame [] in
    Tk.pack [f][Expand true; Fill Fill_Both];
    Applets.call cmo.module_functions f ctx;
    Some (new trivial_display (f, url))
  end
```

⟨*signature* Applets.call 178e⟩≡ (482)
```
val call : (string, applet_callback) Hashtbl.t  ->
           Widget.widget -> Viewers.context -> unit
```

⟨*class* Appview.trivial_display 178f⟩≡ (483d)
```
class trivial_display (w, url) =
 object
  inherit Viewers.display_info ()
  (* val w = w *)
  (* val url = url *)

  method di_widget = w
  method di_abort = ()
  method di_destroy = if Winfo.exists w then Tk.destroy w
  method di_fragment _f = ()
  method di_redisplay = ()
  method di_title = Url.string_of url
  method di_source = ()
  method di_load_images = ()
  method di_update = ()
end
```

⟨Appview.viewer() *in* Loaded *case, if* is_update 179a⟩≡ (178d)
```
  if is_update dh.dh_headers cmo.module_info then begin
    Dload.remove url;
    raise Not_found
  end
```

⟨*signature* Dload.remove 179b⟩≡ (494)
```
  val remove : Url.t -> unit
```

⟨*function* Appview.is_update 179c⟩≡ (483d)
```
  (* Note: since we look in the cache of loaded applets, we must check
   * that this is actually the *same* version of the applet that we
   * are trying to run
   *)
  let is_update this_headers loaded_headers =
    try
      let this_date = Http_headers.get_header "date" this_headers
      and loaded_date = Http_headers.get_header "date" loaded_headers in
       this_date <> loaded_date
    with
      Not_found ->  (* no date means update *)
        true
```

⟨Appview.viewer() *match* Dload.get *cases* 179d⟩+≡ (178a) ◁178d 179e▷
```
  | Rejected old ->
    if is_update dh.dh_headers old then begin
      Dload.remove url;
      raise Not_found
    end
    else
    begin
      Document.dclose true dh;
      Error.f (I18n.sprintf "%s was rejected" (Url.string_of url));
      None
    end
```

⟨Appview.viewer() *match* Dload.get *cases* 179e⟩+≡ (178a) ◁179d
```
  | Unavailable old ->
    if is_update dh.dh_headers old then begin
      Dload.remove url;
      raise Not_found
    end
    else
    begin
      Document.dclose true dh;
      Error.f (I18n.sprintf "%s is not available" (Url.string_of url));
      None
    end
```

16.4.2 Embed viewer

⟨*function* Appview.applet_viewer 179f⟩≡ (483d)
```
  let embed_viewer : Embed.viewer =
   fun _parms frame (ctx : Viewers.context) (doc : Document.t) : unit ->
    let url = doc.document_address in

    let invoke frame ftable =
      (* we need another level of frame where to bind the update event *)
      let f = Frame.create frame [Class "Caml"] in
      Tk.pack [f][Expand true; Fill Fill_Both];

      let caps = Cap.network_caps_UNSAFE () in
      (* if we are asked to update, do it *)
      Frx_synth.bind f "update"
        (fun _ -> Embed.update caps frame ctx doc (fun () -> ()));

      Applets.call ftable f ctx
    in
    try
      (* If the bytecode is loaded, run the thing; else fail *)
      match Dload.get url with
      ⟨Appview.embed_viewer match Dload.get cases 180a⟩
    with
      ⟨Appview.embed_viewer exn handler 182a⟩
```

⟨Appview.embed_viewer *match* Dload.get *cases* 180a⟩≡ (179f) 180b ▷

```
  | Loaded cmo ->
    if is_update doc.document_headers cmo.module_info then begin
      Dload.remove url;
      raise Not_found
    end else
      invoke frame cmo.module_functions
```

⟨Appview.embed_viewer *match* Dload.get *cases* 180b⟩+≡ (179f) ◁ 180a 180e ▷

```
  | Rejected old ->
    if is_update doc.document_headers old then begin
      Dload.remove url;
      raise Not_found
    end else
     Applets.error frame (I18n.sprintf "%s was rejected" (Url.string_of url))
```

⟨*signature* Applets.error 180c⟩≡ (482)

```
  val error : Widget.widget -> string -> unit
```

⟨*function* Applets.error 180d⟩≡ (483a)

```
  (* [error frame errmsg] reports an Error in applet evaluation (that is,
     in evaluation of entry points, since callbacks are on a different
     thread of control). [frame] is the applet frame. *)
  let error frame msg =
    let t = I18n.sprintf "Applet Error: %s" msg in
    Tk.pack[Label.create frame [Text t]] []
```

⟨Appview.embed_viewer *match* Dload.get *cases* 180e⟩+≡ (179f) ◁ 180b

```
  | Unavailable old ->
    if is_update doc.document_headers old then begin
      Dload.remove url;
      raise Not_found
    end else
      Applets.error frame (I18n.sprintf "%s is not available" (Url.string_of url));
```

16.5 Loading OCaml bytecode: Dynlink

16.5.1 mod_cache

⟨*constant* Dload.mod_cache 180f⟩≡ (493)

```
  let mod_cache  : (Url.t, mod_status) Hashtbl.t = Hashtbl_.create ()
```

⟨*function* Dload.remove 181a⟩≡ (493)
```
  let remove = Hashtbl.remove mod_cache
```

⟨*function* Dload.iter 181b⟩≡ (493)
```
  let iter f = Hashtbl.iter f mod_cache
```

⟨*signature* Dload.iter 181c⟩≡ (494)
```
  val iter : (Url.t -> mod_status -> unit) -> unit
```

⟨*function* Dload.get 181d⟩≡ (493)
```
  let get = Hashtbl.find mod_cache
```

⟨Appview.viewer() *exn handler* 181e⟩≡ (178a)
```
  Not_found ->
    let f = Frame.create frame [] in
    (* Otherwise, queue it, and if it's the first request for this applet,
       save the code and loadit when finished *)
    if Dload.add_pending_applet url (fun ftable -> Applets.call ftable f ctx)
    then begin
     let fname = Msys.mktemp "bytc"
     and buffer = Bytes.create 2048 in
     let oc = open_out_bin fname in
     dh.document_feed.feed_schedule
        (fun _ ->
          try
            let n = dh.document_feed.feed_read buffer 0 2048 in
            if n = 0 then begin
             Document.dclose true dh;
             close_out oc;
             let doc = Document.{ document_address = dh.document_id.document_url;
               document_data = FileData(Fpath.v fname,true);
               document_headers = dh.dh_headers} in
             Cache.add dh.document_id doc;
             Cache.finished dh.document_id;

             Dload.load doc

            end else
             output oc buffer 0 n
          with
            Unix.Unix_error(_,_,_) ->
             Document.dclose true dh;
             close_out oc;
             Msys.rm fname;
             Error.f (I18n.sprintf "Error during loading of bytecode")
        );
    end;
    Some (new trivial_display(f, url))
```

⟨*signature* Dload.add_pending_applet 181f⟩≡ (494)
```
  val add_pending_applet :
    Url.t -> ((string, applet_callback) Hashtbl.t -> unit) -> bool
  (* returns true if this is the first applet for this bytecode *)
```

⟨*signature* Dload.load 181g⟩≡ (494)
```
  val load: Document.t  -> unit
```

⟨Appview.embed_viewer *exn handler* 182a⟩≡ (179f)
```
  Not_found ->
    (* Otherwise, queue it, and if it’s the first request for this applet,
      load the bytecode *)
    if Dload.add_pending_applet url (invoke frame)
    then
     Dload.load doc (* This will flush the queue of invocations.
            Since loading is interactive, other calls
            to applet_viewer for the same applet may occur
            concurrently, in which case their invocation
            will be stored in the queue and flushed after
            loading.
            *)
```

16.5.2 Dload.load()

⟨*function* Dload.load 182b⟩≡ (493)
```
  (* Load a foreign bytecode, populate mod_cache *)
  let load (doc : Document.t) : unit  =
    let url = doc.document_address in
    Logs.info (fun m -> m "dynamic loading doc %s" (Url.string_of url));
    (* do we have it already loaded ? TODO: check last modified *)
    try
      ignore (Hashtbl.find mod_cache url) (* then forget it *)
    with
      Not_found ->
       (* actually load it from some file. [remove] says if we should
          remove the file after loading. If the file is from the cache,
          it is NOT our responsability to remove it
        *)
        let file, remove =
          match doc.document_data with
          | FileData(file,_) -> file, false
          | MemoryData buf ->
            let file = Msys.mktemp "mmmbuf" in
            let oc = open_out file in
            output_string oc (Ebuffer.get buf);
            close_out oc;
            Fpath.v file, true
        in
        try
          match applet_kind doc !!file with
          ⟨Dload.load() match applet_kind cases 183b⟩
        with
         Failure "dontkeep" ->
           if remove then Msys.rm !!file;
           Hashtbl.add mod_cache url (Rejected doc.document_headers);
           Error.f (I18n.sprintf "%s was rejected" (Url.string_of url))
```

16.5.3 Applet kind

⟨*type* Dload.applet_kind 182c⟩≡ (493)
```
  type applet_kind =
    | Bytecode of bool (* signed ? *)
    | Source
```

⟨*function* Dload.applet_kind 183a⟩≡ (493)
```
  (* may raise Invalid_HTTP_header e *)
  let applet_kind (doc : Document.t) (file : string) =
    let kind () =
      (* TODO: pass caps *)
      let ic = open_in file
      and buf : bytes = Bytes.create 8 in
      really_input ic buf 0 8;
      close_in ic;
      if Bytes.to_string buf = "Caml1999" then Bytecode false else
      if Bytes.sub_string buf 0 2 = "(*" || Bytes.sub_string buf 0 5 = "open " then Source
      else (* check suffix *)
        match Mstring.get_suffix (Url.string_of doc.document_address) with
        | "ml" -> Source
        | "cmo" | "cma" -> Bytecode false
        | "pgp" | "scmo" -> Bytecode true
        | _ -> (* assume pgp signed... *) Bytecode true
    in
    match Lexheaders.media_type (Http_headers.contenttype doc.document_headers) with
    | ("application", "x-caml-applet"), [] -> (* do magic number trick *)
        kind ()
    | ("application", "x-caml-applet"), l ->
        begin
          try
            match String.lowercase_ascii (List.assoc "encoding" l) with
            | "source" -> Source
            | "signed-bytecode" -> Bytecode true
            | "bytecode" -> Bytecode false
            | _ -> kind ()
         with
           Not_found -> kind ()
      end
    | _, _ ->
        failwith "dontkeep"
```

⟨Dload.load() *match* applet_kind *cases* 183b⟩≡ (182b) 189c ▷
```
 | Bytecode false ->
    if ask url then begin
      unsafe_load doc !!file;
      if remove then Msys.rm !!file;
    end else begin
      if remove then Msys.rm !!file;
      failwith "dontkeep"
    end
```

⟨*function* Dload.ask 183c⟩≡ (493)
```
  let ask (url : Url.t) : bool =
    0 = Frx_dialog.f Widget.default_toplevel (Mstring.gensym "accept")
         "MMM Question"
         (I18n.sprintf  "Unsigned bytecode file %s" (Url.string_of url))
         (Predefined "question") 1
         ["Accept"; "Reject"]
```

16.5.4 Low-level unsafe load

⟨*signature* Dload.in_load 183d⟩≡ (494)
```
  val in_load : bool ref
```

⟨*constant* Dload.in_load 183e⟩≡ (493)
```
  let in_load = ref false
```

⟨*signature* Applets.register 184a⟩≡ (482)
```
  val register : string -> applet_callback -> unit
```

⟨*constant* Dload.register_queue 184b⟩≡ (493)
```
  (* Register queue
     The queue is used to make a separate hashtbl for each loaded bytecode,
     and thus give some "proper" name space to each applet.
     e.g. an applet has toplevel expressions such as
     let _ = Applets.register
     NOTE: register must be protected so that it's available only during
     load time. (and even with this protection, it's subject to race
     condition).
   *)
  let register_queue = Queue.create()
```

⟨*function* Dload.unsafe_load 184c⟩≡ (493)
```
  (* Low-level loading of a foreign bytecode stored in a tmp file *)
  let unsafe_load (doc : Document.t) file =
    if !in_load then Error.f (I18n.sprintf "Already loading a module")
    else begin
      in_load := true;
      let url = doc.document_address in
      ⟨Dload.unsafe_load() set capabilities 187a⟩
      (* prepare the register queue *)
      Queue.clear register_queue;

      try
        (* Dynlink call, finally *)
        Dynlink.loadfile_private file;

        ⟨Dload.unsafe_load() reset capabilities 187b⟩
        let funtable = register_flush() in
        in_load := false;

        Hashtbl.add mod_cache url (Loaded
                     { module_address = Url.string_of url;
                       module_info = doc.document_headers;
                       module_functions = funtable;
                     });

        (* "run" the applets (evaluate continuations that run the entry point) *)
        flush_pending_applets url funtable;
      with e ->
       ⟨Dload.unsafe_load() exn e handler 184d⟩
    end
```

⟨Dload.unsafe_load() *exn* e *handler* 184d⟩≡ (184c)
```
  Capabilities.reset();
  ignore (register_flush());
  in_load := false;
  (match e with
  | Dynlink.Error e ->
      Error.f (I18n.sprintf "Failed to load Caml module %s\n%s"
                           (Url.string_of url) (dynlinkerror e));
      failwith "dontkeep"
  | e ->
      Error.f (I18n.sprintf "Failed to load Caml module %s\n%s"
                            (Url.string_of url)
                            (Printexc.to_string e));
      failwith "dontkeep"
  )
```

⟨*function* Applets.register 185a⟩≡ (483a)
```
(* Oups. This is defined in Dload *)
let register name cb =
  if !Dload.in_load then Dload.register name cb
```

⟨*signature* Dload.register 185b⟩≡ (494)
```
val register : string -> applet_callback -> unit
```

16.5.5 Registered functions

⟨*function* Dload.register 185c⟩≡ (493)
```
(* This is the one that we export to applets *)
let register name f =
  Queue.add (name, f) register_queue
```

⟨*function* Dload.register_flush 185d⟩≡ (493)
```
(* Create a hashtable of functions
   This is called after loading the bytecode
 *)
let register_flush () =
  let names = Hashtbl_.create () in
  try
    while true do
      let (name,f) = Queue.take register_queue in
      Hashtbl.add names name f
    done;
    names
  with
    Queue.Empty -> names
```

16.5.6 Pending loads

⟨*constant* Dload.pending_loads 185e⟩≡ (493)
```
let pending_loads :
  (Url.t, ((string, applet_callback) Hashtbl.t -> unit) Queue.t) Hashtbl.t
   = Hashtbl_.create ()
```

⟨*function* Dload.add_pending_applet 185f⟩≡ (493)
```
(* add to queue while loading
 *   DO NOT CALL THIS if the bytecode is already loaded
 *   The queue will be flushed whenever the bytecode gets loaded
 *)
let add_pending_applet url cont =
    (* we already have a queue of applets waiting for url *)
    if Hashtbl.mem pending_loads url then false else
      (* this is the first request for this Url *)
      let q = Queue.create() in
      Queue.add cont q; (* add continuation *)
      Hashtbl.add pending_loads url q;
      true
```

⟨*function* Dload.flush_pending_applets 185g⟩≡ (493)
```
(* Evaluate all pending continuations for this bytecode *)
let flush_pending_applets url ftable =
  try
    let q = Hashtbl.find pending_loads url in
    Hashtbl.remove pending_loads url;
    try
```

```
      while true do
       (Queue.take q) ftable
      done
    with
      Queue.Empty -> ()
  with
    Not_found -> (* url not in pending_loads table. Is that an error ? *)
      ()
```

16.6 Calling OCaml bytecode

⟨*function* Applets.call 186a⟩≡ (483a)

```
  (* [call table frame context]
     calls the main entry point of an applet
     [table] : table of entry points registered by a bytecode file
     [frame] : applet frame
     [context] : Viewers.context
       viewer_params contains attributes of the EMBED tag
  *)
  let call table frame ctx =
    let fname =
      try
        List.assoc "function" ctx#params
      with
        Not_found -> "main" (* default entry point *) in
    try
      let foo = Hashtbl.find table fname in
      (* destroy the alt window of the frame*)
      List.iter Tk.destroy (Winfo.children frame);
      try
        Printexc.print (foo frame) ctx
      with
        e -> error frame
          (I18n.sprintf "Applet function \"%s\" raised exception: %s"
             fname (Printexc.to_string e))
    with
      Not_found ->
        (* mismatch between EMBED function= and registered entry points *)
        error frame (I18n.sprintf "Applet function \"%s\" not found" fname)
```

16.7 Sandboxing

⟨Appsys.init() *sandbox restrictions* 186b⟩≡ (177b)

```
  (* old: modern OCaml does not need anymore to use this old API
   *  Dynlink.init();
   *  Dynlink.add_available_units Crcs.crc_unit_list;
   *  Dynlink.add_available_units Crcsmmm.crc_unit_list;
  *)
  (* TODO: use instead Dynlink.set_allowed_units ?
   * TODO: allow_only does not seem to work :(
   *)
  Dynlink.allow_only ["Safe419"];
  (* old? needed? pad: TODO, infer using `ocamlc -where`
     Dynlink.add_interfaces ["Pervasives"; "Unix"]
      ["/opt/local/lib/ocaml"];
  *)
```

16.8 Capabilities

⟨Dload.unsafe_load() *set capabilities* 187a⟩≡ (184c)
```
  Capabilities.set
    ((if !paranoid then Capabilities.strict_default url
      else Capabilities.lenient_default url));
```

⟨Dload.unsafe_load() *reset capabilities* 187b⟩≡ (184c)
```
  Capabilities.reset();
```

16.8.1 Paranoid setting

⟨*signature* Dload.paranoid 187c⟩≡ (494)
```
  val paranoid : bool ref
```

⟨*constant* Dload.paranoid 187d⟩≡ (493)
```
  let paranoid = ref true  (* selects default capabilities *)
```

⟨Appsys.applets_pref *other elements* 187e⟩≡ (177d)
```
  Prefs.bool_pref "Paranoid" Dload.paranoid
```

16.9 Advanced features

16.9.1 Applet preferences

16.9.2 Applet user menu

⟨Mmm.navigator() *User menu* 187f⟩≡ (44a)
```
  (* User menu, extensible by applets *)
  let userb = Menubutton.create_named mbar "user" [Text (s_ "User")] in

  let userm = ref (Menu.create_named userb "menu" []) in
  let reset_user_menu _ =
    Tk.destroy !userm;
    userm := Menu.create_named userb "menu" [];
    !user_menus |> List.iter (fun (entry, f) ->
        Menu.add_command !userm
            [Label entry;
             Command (fun () -> f (Nav.make_ctx caps nav hist.h_current.h_did))]
    );
    Menubutton.configure userb [Menu !userm]
  in
  reset_user_menu();
  Frx_synth.bind userb "user_menu" reset_user_menu;
```

⟨*signature* Mmm.add_user_menu 187g⟩≡ (459)
```
  (* Used for applets *)
  val add_user_menu : string -> (Viewers.context -> unit) -> unit
```

⟨*constant* Mmm.user_menus 187h⟩≡ (460)
```
  (* User defined menus *)
  let user_menus = ref []
```

⟨*function* Mmm.add_user_menu 187i⟩≡ (460)
```
  let add_user_menu entry f =
    user_menus := (entry,(fun x -> try f x with _ ->())) :: !user_menus;
    Frx_synth.broadcast "user_menu"
```

16.9.3 Machine hooks

⟨*signature* Html_disp.add_hook 188a⟩≡ (423a)
```
  val add_hook: (machine -> unit) -> unit
```

⟨*function* Html_disp.add_hook 188b⟩≡ (423b)
```
  let add_hook f =
    user_hooks := f :: !user_hooks
```

⟨*constant* Html_disp.user_hooks 188c⟩≡ (423b)
```
  (* Hooks for applets/modules. Control is made elsewhere *)
  let user_hooks = ref []
```

⟨Html_disp.Make.create() *run user hooks* 188d⟩≡ (125b)
```
  !user_hooks |> List.iter (fun f -> f (mach :> machine));
```

16.9.4 External windows

⟨*signature* Applets.get_toplevel_widget 188e⟩≡ (482)
```
  val get_toplevel_widget : Tk.options list -> Widget.widget
```

⟨*function* Applets.get_toplevel_widget 188f⟩≡ (483a)
```
  (* Support for "external" windows *)
  let get_toplevel_widget options =
    Toplevel.create Widget.default_toplevel options
```

16.9.5 Initial modules and mmm -nomodule

⟨Main.main() *locals* 188g⟩+≡ (29c) ◁117b 207i▷
```
  let modules = ref true in
```

⟨Main.main() *command line options* 188h⟩+≡ (29c) ◁117c 208a▷
```
  "-nomodule", Arg.Unit (fun () -> modules := false),
  " Don't load initial modules";
```

⟨Appsys.init() *load local applets* 188i⟩≡ (177b)
```
  (* Load local applets *)
  if initialp then load_initial_modules();
```

⟨*function* Appsys.load_initial_modules 188j⟩≡ (483b)
```
  (* Load "initial modules" residing in user directory (~/.mmm) *)
  let load_initial_modules () =
    try
      let dir =
        Filename.concat (Filename.concat (Sys.getenv "HOME") ".mmm")
                        (string_of_int Version.number) in
      if Sys.file_exists dir then
      let dh = Unix.opendir dir in
      try
        while true do
          let f = Unix.readdir dh in
          if Filename.check_suffix f ".cmo" then
            Dload.load_local (Filename.concat dir f)
          done
      with
       | End_of_file -> Unix.closedir dh
       | e -> Unix.closedir dh; raise e
    with
    | Not_found (* Sys.getenv *) ->
```

```
        prerr_endline "Please specify the HOME environment variable";
        raise (Exit.ExitCode (-1))
    | Unix.Unix_error (e, fname, arg) ->
        Error.f (I18n.sprintf
           "Error during loading of initial modules\n%s: %s %s"
           fname (Unix.error_message e) arg)
```

⟨*signature* Dload.load_local 189a⟩≡ (494)

```
  val load_local : string -> unit
```

⟨*function* Dload.load_local 189b⟩≡ (493)

```
  (* Loading of local extensions *)
  let load_local (file : string) =
    Logs.info (fun m -> m "loading local mmm extension %s" file);
    if !in_load then Error.f (I18n.sprintf "Already loading a module")
    else begin
     in_load := true;
     let url = Lexurl.make ("file://"^file) in
     Capabilities.set (Capabilities.local_default url);
     (* prepare the register queue *)
     Queue.clear register_queue; (* this is not needed actually *)
     try
       Dynlink.loadfile_private file;
       Capabilities.reset();
       let funtable = register_flush() in
       in_load := false;
       Hashtbl.add mod_cache url (Loaded
                    { module_address = Url.string_of url;
                      module_info = [];
                      module_functions = funtable;
                    });
       (* "run" the applets (evaluate continuations that run the entry point) *)
       flush_pending_applets url funtable;
     with e ->
       (* In case of any error here *)
       Capabilities.reset();
       ignore (register_flush());
       in_load := false;
       match e with
       | Dynlink.Error e ->
         Error.f (I18n.sprintf "Failed to load Caml module %s\n%s"
                                 (Url.string_of url) (dynlinkerror e))
       | e ->
         Error.f (I18n.sprintf "Failed to load Caml module %s\n%s"
                                 (Url.string_of url)
                                 (Printexc.to_string e))
    end
```

16.9.6 Loading OCaml Source

⟨Dload.load() *match* applet_kind *cases* 189c⟩+≡ (182b) ◁183b 190a▷

```
  | Source ->
    (* the mmmc script is part of the distribution *)
    let byt = Msys.mktemp "apcode" in
    let cmd = spf "mmmc %s %s" !!file byt in
    begin match Sys.command cmd with
    | 0 ->
        unsafe_load doc byt;
        Msys.rm byt;
        if remove then Msys.rm !!file
    | _n ->
      Error.f (I18n.sprintf "Can't compile applet %s"
              (Url.string_of url));
      Msys.rm byt;
      if remove then Msys.rm !!file
   end
```

16.9.7 Loading signed bytecode

⟨Dload.load() *match* applet_kind *cases* 190a⟩+≡ (182b) ◁189c

```
  | Bytecode true ->
    begin
     match Pgp.check (Url.string_of url) !!file with
     | Some clear ->
       unsafe_load doc clear;
       Msys.rm clear;
       if remove then Msys.rm !!file
     | None -> (* was refused or malformed *)
       if remove then Msys.rm !!file;
       failwith "dontkeep"
   end
```

⟨*toplevel comment* Pgp 190b⟩≡ (495)

```
  (*
     decoding goes as follows:
      pgp -f +batchmode < signed > unsigned 2> log
     to produce a signed file,
      pgp -s foo # produces foo.pgp
   or pgp -sta foo # produces foo.asc

  pgp_check is called on file (saved by a scheduler)
  when Content-Encoding was specified as PGP
  *)
```

⟨*function* Pgp.check 190c⟩≡ (495)

```
  let check url signed_file =
   let clear_file = Msys.mktemp "clear" in
     let rec attempt () =
       let (lin, lout) = Unix.pipe() in
         match Low.fork() with
           0 -> Unix.close lin;
            batch_pgp signed_file clear_file lout
                 (* never reached *)
            (* old: None *)
         | n ->
            Unix.close lout;
        let res = read_all lin in
        Unix.close lin;
        let _p, st = Unix.waitpid [] n in
        match st with
            WEXITED n ->
               begin match
            Frx_dialog.f Widget.default_toplevel (Mstring.gensym "pgp")
              "PGP Authentification"
                     (I18n.sprintf "PGP diagnostic for %s\n%s" url res)
              (Predefined "question")
              (if n = 0 then 0 else 1)
              ["Accept"; "Reject"; "Retry"] with
              |  0 -> (* yeah *) Some clear_file
         | 1 -> (* duh *) Msys.rm clear_file; None
         | 2 -> attempt ()
                |  _ -> assert false
         end
        | _ ->
          Msys.rm clear_file;
             Error.f "PGP aborted";
          None
  in
  attempt()
```

⟨*function* Pgp.batch_pgp 191a⟩≡ (495)

```
  let batch_pgp signed clear pgplog  =
    let oin = Unix.openfile signed [O_RDONLY] 0
    and oout = Unix.openfile clear  [O_WRONLY; O_CREAT] 0o600 in
     Unix.dup2 oin Unix.stdin; Unix.close oin;
     Unix.dup2 oout Unix.stdout; Unix.close oout;
     Unix.dup2 pgplog Unix.stderr; Unix.close pgplog;
    try
      Unix.execvp "pgp" [| "pgp"; "+batchmode=on"; "+verbose=0"; "-f" |]
    with
      Unix.Unix_error(e, _, _) ->
        Printf.eprintf "%s\n" (Unix.error_message e);
        flush Stdlib.stderr;
        raise (Exit.ExitCode 1)
```

⟨*function* Pgp.read_all 191b⟩≡ (495)

```
  (* Read on a channel until eof *)
  let read_all chan =
    let len = 1024 in
    let rec read_chunk (buffer : bytes) (offs : int) : string =
      let chunk = Unix.read chan buffer offs len in
      if chunk = 0 then Bytes.sub_string buffer 0 offs else
        let newoffs = offs + chunk
        and l = Bytes.length buffer in
          if newoffs + len > l then
            let newbuf = Bytes.create (2*l+len) in
             Bytes.unsafe_blit buffer 0 newbuf 0 newoffs;
             read_chunk newbuf newoffs
          else
             read_chunk buffer newoffs
    in
    read_chunk (Bytes.create 2048) 0
```

Chapter 17

Web-developer tools

17.1 View source

⟨*Document menu elements* 192a⟩+≡ (46d) ◁169b 204b▷

```
  [Label (s_ "View Source")    ; Command view_source];
```

⟨*function* Mmm.navigator.view_source 192b⟩≡ (35a)

```
  let view_source () =
    !current_di |> Option.iter (fun di -> di#di_source)
  in
```

⟨Viewers.display_info *other virtual methods signatures* 192c⟩≡ (25b)

```
  method virtual di_source : unit          (* source viewer *)
```

⟨Plain.plain *other methods or fields* 192d⟩≡ (113d)

```
  method di_source = ()
```

⟨Htmlw.display_html *other methods or fields* 192e⟩+≡ (122c) ◁159h

```
  method di_source = self#source

  (* The source is attached to this frame so we can destroy the interior widgets*)
  method source =
    if pending
    then Error.f (s_ "Cannot view document source (pending)")
    else Source.view frame did (fun () -> self#redisplay) errors annotations
           feed_read#get_code
```

⟨*signature* Source.view 192f⟩≡ (370b)

```
  val view:
    Widget.widget ->
    Document.id ->
    (unit -> unit) ->
    (Html.location * string) list ref ->
    (Tk.textTag * Html.location) list ref ->
    Charset.detected_code ->
    unit
```

⟨*function* Source.view 192g⟩≡ (370d)

```
  (* Commit modifies the cache *)
  let view attach did redisplay errors annotations _coding =
    try
      let doc = Cache.find did in
      (* load : take document from cache and put it in text widget
         commit : take source of text widget and store in cache
         save : save to original URL. supported only on file:, could be
         supported on http: with put ?
 *)
let load, cachesave, saveurl =
  match doc.document_data with
  | FileData (fname,_) ->
      let tmpfile = Msys.mktemp "buf" in
      (* load *)
      (fun t ->
         let ic = open_in !!fname in
         let buf = Bytes.create 2048 in
         try
           while true do
             let n = input ic buf 0 2048 in
             if n = 0
             then raise End_of_file
             else Text.insert t textEnd
                    (if n = 2048 then Bytes.to_string buf else Bytes.sub_string buf 0 n) []
           done
         with End_of_file -> close_in ic
      ),

     (* commit *)
     (fun t ->
        let oc = open_out tmpfile in
        output_string oc
        (Text.get t (TextIndex(LineChar(0,0), [])) textEnd);
        close_out oc;
        (* SWITCH CACHE *)
        doc.document_data <- FileData(Fpath.v tmpfile, true)),

    (* save *)
    Some (fun t ->
      let oc = open_out !!fname in
      output_string oc
        (Text.get t (TextIndex(LineChar(0,0), [])) textEnd);
      close_out oc;
      (* SWITCH CACHE *)
      doc.document_data <- FileData(fname, false)
   )

  | MemoryData buf ->
      (* load *)
      (fun t -> Text.insert t textEnd (Ebuffer.get buf) []),
      (* commit *)
      (fun t ->
         Ebuffer.reset buf;
         Ebuffer.output_string buf
          (Text.get t (TextIndex(LineChar(0,0), [])) textEnd)),
      None
in
let top = Toplevel.create attach [Class "MMMSource"] in
Wm.title_set top "HTML source display";

let errorv = Textvariable.create_temporary top in
let f, t =
  new_scrollable_text top [Foreground Black; Background White] false in
let f' = Frame.create_named top "buttons" [] in
let dismiss = Button.create_named f' "dismiss"
  [Text (s_ "Dismiss"); Command (fun _ -> destroy top)] in
let commit = Button.create_named f' "commit" [Text (s_ "Commit")] in
```

```
  let save = Button.create_named f' "save" [Text (s_ "Save")] in
  let err = Button.create_named f' "errors" [] in
  let ferr = Frame.create top [] in
  let err_msg =
    Label.create_named ferr "error"
          [Relief Sunken;TextVariable errorv; Anchor W]
in

(* Error display and looping *)
let error_idx = ref [] in
let get_msg idx =
  let rec f = function
      [] -> raise Not_found
    | (s,e,msg)::l ->
      if Text.compare t s LE idx && Text.compare t idx LE e then msg
   else f l in
  f !error_idx in
let show_error = Text.yview_index t in
(* alternative is : Text.see t but is less practical *)
let loop_in_errors  =
  let current = ref None in
  (fun () ->
    match !current with
     None -> (* select the first error *)
    let (s,e,_) = List.hd !error_idx in
     current := Some e;
     show_error s
    | Some s -> (* select the next one *)
      try
       let (s,e) = Text.tag_nextrange t "errors" s textEnd in
     current := Some (TextIndex (e,[]));
     show_error (TextIndex(s,[]))
   with _ -> (* no more *)
     let (s,e,_) = List.hd !error_idx in
      current := Some e;
      show_error s) in

let mark_errors () =
  error_idx := [];
  List.iter (fun (Loc(s,e),msg) ->
    let idxs = abs_index s
    and idxe = abs_index e in
    Text.tag_add t "errors" idxs idxe;
    error_idx := (idxs, idxe, msg) :: !error_idx)
      !errors;
  match List.length !errors with
    0 ->
     Button.configure err [Text (s_ "No Errors"); State Disabled]
  | _n ->
  Button.configure err
      [Text (s_ "%d errors" (List.length !errors));
       State Normal; Command loop_in_errors]
and decorate = annotate t
in
let reset () =
  (* if we delete the tag, we delete the bindings.
   * we can't use the old indexes since the buffer might have changed ! *)
   let rec remall = function
      [] -> ()
    | [_x] -> ()
    | s::e::l ->
     Text.tag_remove t "errors" (TextIndex(s,[])) (TextIndex(e,[]));
     remall l
    in
  remall (Text.tag_ranges t "errors");
  errors := [];
  Button.configure err
     [Text (s_ "Display Errors"); Command mark_errors; State Normal]

in
   Button.configure commit
      [Command (fun () -> reset(); cachesave t; redisplay())];
   (match saveurl with
      None -> Button.configure save [State Disabled]
    | Some f ->
        Button.configure save
        [Command (fun () -> reset(); f t; redisplay())]);
   Button.configure err
     [Text (s_ "Display Errors");
      Command (fun () -> mark_errors(); decorate !annotations);
      State Normal];
   Text.configure t [Background (NamedColor "white")];
   Text.tag_configure t "errors" [Underline true];
   Text.tag_bind t "errors" [[], Enter]
     (BindSet ([Ev_MouseX; Ev_MouseY],
   (fun ei ->
      (* The index of the click position *)
      let i = Text.index t
          (TextIndex (AtXY (ei.ev_MouseX, ei.ev_MouseY), [])) in
        try
      Textvariable.set errorv (get_msg (TextIndex(i,[])))
        with
      Not_found -> ())));
   Text.tag_bind t "errors" [[], Leave]
     (BindSet ([], (fun _ei -> Textvariable.set errorv "")));

    pack [dismiss;commit;save;err][Side Side_Left];
    pack [err_msg] [Side Side_Left; Expand true; Fill Fill_X];
  pack [ferr][Side Side_Top; Fill Fill_X];
  pack [f'][Side Side_Top; Fill Fill_X];
  pack [f][Fill Fill_Both; Expand true; Side Side_Bottom];

Frx_text.addsearch t;
load t

with Not_found ->
  Error.f "document not in cache"
```

Chapter 18

Concurrency

18.1 Tasks

⟨*signature* `Low.add_task` 196a⟩≡ (275h)
```
  val add_task : (unit -> unit) -> unit
    (* regular tasks *)
```

⟨*constant* `Low.tasks` 196b⟩≡ (277g)
```
  (* There is only a default task *)
  let tasks = ref [
    (fun () -> !cur_tachy#report_traffic tick_duration !bytes_read !sample_read)
    ]
```

⟨*function* `Low.refresh` 196c⟩≡ (277g)
```
  let rec refresh() =
    incr global_time;
    List.iter (fun f -> f ()) !tasks;
    sample_read := 0;
    Timer_.add tick_duration refresh
```

⟨*signature* `Low.global_time` 196d⟩≡ (275h)
```
  val global_time : int ref
```

⟨*constant* `Low.global_time` 196e⟩≡ (277g)
```
  let global_time = ref 0
```

⟨*signature* `Low.init` 196f⟩≡ (275h)
```
  val init : unit -> unit
```

⟨*function* `Low.init` 196g⟩≡ (277g)
```
  let init () = refresh ()
```

18.2 Tachymeter

18.2.1 backend

⟨*global* `Low.bytes_read` 196h⟩≡ (277g)
```
  let bytes_read = ref 0
```

⟨*global* `Low.sample_read` 196i⟩≡ (277g)
```
  let sample_read = ref 0
```

⟨*signature* `Low.cur_tachy` 196j⟩≡ (275h)
```
  val cur_tachy : tachymeter ref
```

18.2.2 frontend

⟨Mmm.navigator() *set tachymeter* 197a⟩≡ (41a)
```
  (* put this as a function so we can restart it if needed *)
  let rec restart_tachy () =
     (* We must not pass hgroup to tachymeter applets *)
     let fcontainer = Frame.create hgroup [] in
     container_frame := Some fcontainer;
     (* restart it if destroyed *)
     bind fcontainer [[], Destroy] (BindSet ([Ev_Widget], (fun ei ->
        if ei.ev_Widget = fcontainer
           && Winfo.exists hgroup (* but we're not dead *)
        then restart_tachy()))
     );
    let rw = Winfo.reqwidth fcontainer in
    let rh = Winfo.reqheight fcontainer in
    Wm.minsize_set top rw rh;
    pack [fcontainer][Side Side_Right; Anchor N];

    start_tachy();

    (* Bad hack to do bindings for our own internal tachymeter:
     * others, in applets, can just access these functions from the safe
     * library
     *)
    if !tachy_maker == About.create_tachy then begin
      match Winfo.children fcontainer with
      | [c] ->
          bind c (Glevents.get "tachy_new") (BindSet ([], (fun _ ->
            new_window_initial ())));
          bind c (Glevents.get "tachy_sel") (BindSet ([], (fun _ ->
            new_window_sel ())));
      | _ -> ()
    end
  in

  restart_tachy(); (* first initialisation *)

  (* good size for keeping only the tachy *)
  Wm.minsize_set top 80 80;
```

⟨*signature* Mmm.change_tachy 197b⟩≡ (459)
```
  val change_tachy : (Widget.widget -> Low.tachymeter) -> unit
```

⟨*function* Mmm.change_tachy 197c⟩≡ (460)
```
  let change_tachy (t : Widget.widget -> Low.tachymeter) =
    !Low.cur_tachy#quit;
    tachy_maker := t;
    (match !container_frame with
     | Some f ->
        List.iter Tk.destroy (Winfo.children f);
        Low.cur_tachy := t f
     | None -> ()
    )
```

⟨*constant* Mmm.tachy_maker 197d⟩≡ (460)
```
  let tachy_maker = ref About.create_tachy
```

⟨*function* Mmm.start_tachy 197e⟩≡ (460)
```
  let start_tachy () =
    !container_frame |> Option.iter (fun f -> Low.cur_tachy := !tachy_maker f)
```

⟨*constant* Mmm.container_frame 198⟩≡ (460)

```
(* Tachymeter support
 * [container_frame] is the parent frame for displaying a tachymeter
 * It’s initialized only after the first navigator window is created.
 * [tachy_maker] contains the current tachymeter creation function.
 * [change_tachy] changes the current tachymeter. It has immediate
 * effect if the first navigator window is already available. Otherwise,
 * it will take effect at creation time, using [start_tachy].
 *)
let container_frame = ref None
```

Chapter 19

Extra features

19.1 Fragment navigation

⟨Document.handle *other fields* 199a⟩+≡ (22b) ◁105b 229b▷
```
  document_fragment : string option;
    (* fragment (#foo) if any *)
```

⟨Mmm.navigator.show_current() *goto fragment* 199b⟩≡ (38a)
```
  (* bogus if two views with fragment on the same pending document *)
  di#di_fragment frag;
```

⟨Viewers.display_info *fragment virtual method signature* 199c⟩≡ (25b)
```
  method virtual di_fragment : string option -> unit (* for # URIs *)
```

⟨Plain.plain *empty methods* 199d⟩+≡ (113d) ◁169e
```
  method di_fragment _frag = ()
```

⟨Htmlw.display_html *fragment methods* 199e⟩≡ (122c)
```
  method di_fragment =
    mach#see_frag
```

⟨Htmlw.display_html *in feed,* End_of_file *exn handler, goto fragment* 199f⟩≡ (124b)
```
  mach#see_frag dh.document_fragment;
```

⟨Html_disp.machine *fragment method* 199g⟩≡ (27b)
```
  method virtual see_frag : string option -> unit
```

⟨Html_disp.display_machine *fragment methods* 199h⟩≡ (125c)
```
  (* This is an intrusion of graphics, but I don't see any other way
   * The last formatter always tries see_frag...
   *)
  method see_frag = see_frag
```

⟨Html_disp.display_machine *private fields* 199i⟩+≡ (125c) ◁127e
```
  val mutable see_frag = (fun _ -> ())
```

⟨Htmlfmt.formatter *fragment method* 199j⟩≡ (26e)
```
  see_frag : string option -> unit;
```

⟨*function* Textfw_fo.create.see_frag 200a⟩≡ (129c)
```
  (* we try to remember the last "reading" position, so you can easily
   * switch back from a goto to some particular place in the document
   *)
  see_frag = begin
    let prev_frag = ref false in
    let view_mem = ref 0.0 in
    match spec with
    | TopFormatter true -> (* this is pscrolling mode *)
      (function
       | None -> (* no place in particular *)
           if !prev_frag then begin
             try Canvas.yview (Winfo.parent thtml) (MoveTo !view_mem)
             with Protocol.TkError _ -> ()
           end;
           prev_frag := false
       | Some s ->
           if not !prev_frag then begin
             try view_mem := fst (Canvas.yview_get (Winfo.parent thtml))
             with Protocol.TkError _ -> ()
          end;
          prev_frag := true;
          if s <> "" then
            try
              let _,y,_,_,_  = Text.dlineinfo thtml
                   (TextIndex (Mark ("#"^s), [LineOffset (-2)]))
              and _,ye,_,_,_ = Text.dlineinfo thtml
                   (TextIndex (End, [CharOffset (-1)])) in
             Canvas.yview (Winfo.parent thtml)
                  (MoveTo (float y /. float ye))
            with Protocol.TkError _ -> ()
     )
    | _ ->
      (function
      | None -> (* no place in particular *)
          if !prev_frag then begin
            (* we were at view_mem *)
            try Text.yview thtml (MoveTo !view_mem)
            with Protocol.TkError _ -> ()
          end;
          prev_frag := false
      | Some s -> (* go to s *)
          if not !prev_frag then begin
            (* we were not in some special place, remember it *)
            try view_mem := fst (Text.yview_get thtml)
            with Protocol.TkError _ -> ()
          end;
          prev_frag := true;
          if s <> "" then
            try Text.yview_index thtml
               (TextIndex (Mark ("#"^s), [LineOffset (-1)]))
            with Protocol.TkError _ -> ()
      )
  end
```

19.2 History navigation

⟨*type* History.t 200b⟩≡ (449c 448b)

```
  type t = {
    mutable h_start : entry;
    mutable h_current: entry;

    h_key : int;
    mutable h_first : bool
    }
```

⟨*type* History.history_entry 201a⟩≡ (449c 448b)

```
  (*
     Linear history: we keep going adding to the end of the list,
     EXCEPT when you go back and then on a new link.
  *)
  type entry = {
    h_did : Document.id;
    h_fragment : string option;

    h_prev : entry option;
    mutable h_next : entry option
    }
```

⟨*constant* History.create 201b⟩≡ (449c)

```
  let create =
    let keycnter = ref 0 in
    (fun did ->
      incr keycnter;
      let e = { h_did = did;
                h_fragment = None;
                h_prev = None;
                h_next = None } in
      { h_key = !keycnter;
        h_start = e;
        h_current = e;
        h_first = true
      })
```

⟨*function* History.back 201c⟩≡ (449c)

```
  let back h =
    match h.h_current.h_prev with
    | None -> None
    | Some e -> h.h_current <- e; Some (e.h_did, e.h_fragment)
```

⟨*function* History.forward 201d⟩≡ (449c)

```
  let forward h =
    match h.h_current.h_next with
    | None -> None
    | Some e -> h.h_current <-e ; Some (e.h_did, e.h_fragment)
```

⟨*signature* Nav.historygoto 201e⟩≡ (450f)

```
  val historygoto : < Cap.network; ..> ->
    t -> Document.id -> string option -> bool -> bool
```

⟨*function* Nav.historygoto 201f⟩≡ (453)

```
  (* Used by navigators for back/forward/reload *)
  let historygoto (caps : < Cap.network; ..>)
       (nav : t) (did : Document.id) frag (usecache : bool) : bool =
    Logs.debug (fun m -> m "historygoto");
    if did.document_stamp = Document.no_stamp then begin
      (* we can safely consider this as normal navigation *)
      let uri : string =
        match frag with
        | None -> Url.string_of did.document_url
        | Some f -> sprintf "%s#%s" (Url.string_of did.document_url) f
      in
      (* modify wr *)
      let follow_link (lk : Hyper.link) =
        lk |> request caps nav
          (process_viewer false (make_ctx caps)) (* don't add to history *)
          (usecache,
           (fun wr ->
              if not usecache
              then wr.www_headers <- "Pragma: no-cache" :: wr.www_headers;
              wr),
           specific_viewer false)
      in
      follow_link (Hyper.default_link uri);
      true
    end else begin
      (* the url is a "non-unique" document, that is, its url is not
       * enough to retrieve the document. We should not attempt to
       * reload or retrieve if flushed from the cache
       *)
      try
        let di = Gcache.find nav.nav_id did in
        nav.nav_show_current di frag;
        true
      with Not_found ->
        nav.nav_error#f (s_ "Document was flushed from cache, and should be reloaded from its url\n(probably a PO
        false
     end
```

19.3 Lifecycle control

19.3.1 Abort

⟨Mmm.navigator() *navigation buttons* 202a⟩+≡ (41a) ◁47g 203g▷
```
  let abortb = Button.create_named buttons
    "abort" [Text (s_ "Abort"); Command abort] in
```

⟨*type* Www.aborter 202b⟩≡ (291a 290b)
```
  type aborter = unit -> unit
```

⟨*function* Mmm.navigator.abort 202c⟩≡ (35a)
```
  let abort () =
    actives |> Hashtbl.iter (fun _url aborter ->
      aborter()
    );
    Hashtbl.clear actives;
    !current_di |> Option.iter (fun di -> di#di_abort)
  in
```

⟨*local* Mmm.navigator.actives 202d⟩≡ (34c)
```
  let actives : (Url.t, Www.aborter) Hashtbl.t = Hashtbl_.create () in
```

⟨Nav.t *manage active connections methods* 202e⟩≡ (35b)
```
  nav_add_active : Url.t -> Www.aborter -> unit;
  nav_rem_active : Url.t -> unit;
```

⟨Mmm.navigator() *set nav fields* 203a⟩+≡ (34a) ◁48d 207a▷
```
  nav_add_active = (fun url aborter -> Hashtbl.add actives url aborter);
  nav_rem_active = (fun url -> Hashtbl.remove actives url);
```

⟨Viewers.display_info *lifecycle virtual methods signatures* 203b⟩≡ (25b) 204c▷
```
  method virtual di_abort : unit   (* stop display *)
```

⟨Plain.plain *private fields* 203c⟩+≡ (113d) ◁114b
```
  val mutable (*private*) terminated = false
```

⟨Plain.plain *abort methods* 203d⟩≡ (113d)
```
  method di_abort =
    self#finish true

  (* [finish abort?] *)
  method finish abort =
    if not terminated then begin
      terminated <- true;
      self#ctx#log (if abort then "Aborted" else "");
      Document.dclose true dh;
    end
```

⟨Htmlw.display_html *private fields* 203e⟩+≡ (122c) ◁160a 205b▷
```
  val mutable (*private*) terminated = false
```

⟨Htmlw.display_html *abort methods* 203f⟩≡ (122c)
```
  method di_abort =
    self#finish true

  (* [finish abort?] *)
  method finish abort =
    if not terminated then begin
      terminated <- true;
      ctx#log (if abort then "Aborted" else "");
      Document.dclose true dh
    end;
    (* This has to happen even if we already finished displaying the document *)
    if abort then begin
      Img.ImageScheduler.stop dh.document_id;
      Embed.EmbeddedScheduler.stop dh.document_id
      (* TODO we should also require embedded objects to abort *)
    end
```

19.3.2 Refresh

⟨Mmm.navigator() *navigation buttons* 203g⟩+≡ (41a) ◁202a
```
  let reloadb = Button.create_named buttons
    "reload" [Text (s_ "Reload"); Command reload] in
```

⟨*function* Mmm.navigator.reload 203h⟩≡ (35a)
```
  let reload () =
    let did = hist.h_current.h_did in
    let frag = hist.h_current.h_fragment in
    if did.document_stamp = Document.no_stamp then begin
      (* kill both in cache and in gcache *)
      Cache.kill did;
      Gcache.remove hist.h_key did;
      Nav.historygoto caps nav did frag false |> ignore
    end
    else error#f
        (s_ "Document cannot be reloaded from its url\n(probably a POST request)")
  in
```

⟨*function* `Mmm.navigator.update` 204a⟩≡ (35a)
```
  let update (nocache : bool) =
    let did = hist.h_current.h_did in
    if did.document_stamp = Document.no_stamp then
      Nav.update caps nav did nocache
    else (* POST result *)
      error#f (s_ "Can't update document\n(probably a POST request)")
  in
```

19.3.3 Redisplay

⟨*Document menu elements* 204b⟩+≡ (46d) ◁192a 205e▷
```
  [Label (s_ "Redisplay")      ; Command redisplay];
```

⟨`Viewers.display_info` *lifecycle virtual methods signatures* 204c⟩+≡ (25b) ◁203b
```
  method virtual di_redisplay : unit  (* redisplay *)
```

⟨*function* `Mmm.navigator.redisplay` 204d⟩≡ (35a)
```
  let redisplay () =
    !current_di |> Option.iter (fun di -> di#di_redisplay)
  in
```

⟨`Plain.plain` *redisplay methods* 204e⟩≡ (113d)
```
  method di_redisplay =
    self#redisplay

  (* to redisplay, we have to destroy all widgets, then restart, except
     that we don't use the feed, but rather the cache *)
  method redisplay =
    failwith "redisplay:TODO"
    (*
    try
      dh <- Decoders.insert (Cache.renew_handle dh);
      Winfo.children frame |> List.iter destroy;
      self#init
    with Not_found ->
      Error.f (s_ "Document not in cache anymore")
    *)
```

⟨`Htmlw.display_html` *redisplay methods* 204f⟩≡ (122c)
```
  method di_redisplay =
    self#redisplay

  (* to redisplay, we have to destroy all widgets, then restart, except
     that we don't use the initial feed, but rather the cache *)
  method redisplay =
    if pending
    then Error.f (s_ "Cannot redisplay document (pending)")
    else
      try
        dh <- Decoders.insert (Cache.renew_handle dh);
        Winfo.children frame |> List.iter destroy;
        self#init init_mode
      with Not_found ->
        Error.f (s_ "Document not in cache anymore")
```

19.4 Progress feedback

⟨Plain.plain *progress methods* 205a⟩≡ (113d)
```
  (* progress report *)
  val mutable set_progress = Progress.no_meter
  method set_progress = set_progress
```

⟨Htmlw.display_html *private fields* 205b⟩+≡ (122c) ◁203e
```
  val mutable set_progress = Progress.no_meter
```

⟨Htmlw.display_html *progress method* 205c⟩≡ (122c)
```
  (* progress report *)
  method set_progress = set_progress
```

⟨*function* Htmlw.progress_report 205d⟩≡ (391b)
```
  (* Builds the progress report and pointsto zone.
   * Adds ctx nav function for pointsto
   *)
  let progress_report top ctx =
    let f = Frame.create_named top "progress" [] in
    let pointstov = Textvariable.create_temporary f in
    let pointsto = Textvariable.set pointstov in
    let lpoint =
      Label.create_named f "pointsto" [TextVariable pointstov; Anchor W] in
    let fprog =
      Frame.create_named f "fr" [Width (Pixels 200); Height(Pixels 5)]in

    (* progress meter requires an alt widget, but we don't have to pack it *)
    let _fakealt = Label.create_named fprog "alt" [] in
    pack [fprog][Side Side_Left];
    pack [lpoint][Side Side_Left; Fill Fill_X];
    (* hack to avoid lpoint forcing the navigator to grow like hell *)
    Frame.configure f
      [ Width (Pixels (Winfo.reqwidth (Winfo.toplevel f)));
        Height (Pixels (Winfo.reqheight lpoint))];
    Pack.propagate_set f false;

    ctx#add_nav ("pointsto" ,
                 Viewers.{ hyper_visible = false;
                   hyper_title = "Show target";
                   hyper_func = (fun _ h ->
                     let target =
                       try Hyper.string_of h
                       with Hyper.Invalid_link _msg -> "invalid link"
                     in
                     pointsto target
                   )});
    ctx#add_nav ("clearpointsto" ,
                 { hyper_visible = false;
                   hyper_title = "Clear target";
                   hyper_func = (fun _ _h -> pointsto "")
                 });
    f, Tk_progress.meter fprog
```

19.5 Bookmarks

19.6 Hotlist

⟨*Document menu elements* 205e⟩+≡ (46d) ◁204b

```
  [Label (s_ "Add to hotlist") ; Command add_to_hotlist];
```

⟨*constant* Mmm.hotlist 206a⟩≡ (460)
```
  (* Preference settings *)
  let _hotlist = ref ""
```

⟨*function* Mmm.navigator.add_to_hotlist 206b⟩≡ (35a)
```
  let add_to_hotlist () =
    match !current_di with
    | None -> ()
    | Some di ->
        Hotlist.f (Url.string_of hist.h_current.h_did.document_url) di#di_title
  in
```

⟨*constant* Hotlist.program 206c⟩≡ (285d)
```
  (* A cool module *)
  let program = ref ""
```

⟨*function* Hotlist.f 206d⟩≡ (285d)
```
  let f url title =
    match !program with
    | "" -> Error.f (s_ "No hotlist command defined")
    | s ->
        let _ = Munix.system_eval s ["URL", url; "TITLE", title] true in
        Error.ok (s_ "%s\nadded to hotlist with title\n%s" url title)
```

19.7 Multiple windows

⟨*signature* Mmm.main_navigator 206e⟩≡ (459)
```
  val main_navigator : Nav.t option ref
```

⟨*constant* Mmm.main_navigator 206f⟩≡ (460)
```
  let main_navigator = ref None
```

⟨*constant* Mmm.navigators 206g⟩≡ (460)
```
  (*
   * A navigator window
   *)
  let navigators = ref 0
```

⟨Mmm.navigator() *new navigator hook* 206h⟩≡ (34a)
```
  incr navigators;
```

⟨Mmm.navigator() *destroy navigator hook* 206i⟩≡ (42b) 237h ▷
```
  decr navigators;
```

⟨Mmm.navigator() *exn handler, when multiple navigators* 206j⟩≡ (34a)
```
  else begin
    Tk.destroy top;
    None
  end
```

⟨Nav.t *other fields* 206k⟩≡ (35b)
```
  nav_new : Hyper.link -> unit;
```

⟨Mmm.navigator() *set nav fields* 207a⟩+≡ (34a) ◁203a 238e▷

```
  nav_new = (fun link ->
     try
       let wwwr = Plink.make link in
       navigator caps false wwwr.www_url |> ignore
     with Hyper.Invalid_link _msg ->
       error#f (s_ "Invalid link")
  );
```

⟨*signature* Mmm.new_window_initial 207b⟩≡ (459)

```
  (* ?? *)
  val new_window_initial : < Cap.network; ..> -> unit
```

⟨*signature* Mmm.new_window_sel 207c⟩≡ (459)

```
  (* ?? *)
  val new_window_sel : < Cap.network; ..> -> unit
```

⟨*function* Mmm.new_window_initial 207d⟩≡ (460)

```
  and new_window_initial (caps: < Cap.network; ..>) =
   navigator caps false
     (match !initial_page with | Some u -> u | None -> assert false) |> ignore
```

⟨*function* Mmm.new_window_set 207e⟩≡ (460)

```
  and new_window_sel (caps: < Cap.network; ..>) =
    try
      let url = Selection.get [] in
      navigator caps false (Lexurl.make url) |> ignore
    with _ -> new_window_initial caps
```

⟨Mmm.navigator() *nested functions* 207f⟩+≡ (34a) ◁35a

```
  let new_window () =
       navigator caps false hist.h_current.h_did.document_url |> ignore
  in
  let new_window_initial () =
       navigator caps false initial_url |> ignore
  in
  let new_window_sel () =
    try
       let url = Selection.get [] in
       navigator caps false (Lexurl.make url) |> ignore
    with _ -> navigator caps false initial_url |> ignore
  in
```

19.8 User preferences

19.8.1 Preference file, `mmm -prefs`

⟨*constant* Mmm.preferences 207g⟩≡ (460)

```
  (* placeholder for preference panel *)
  let preferences = ref (fun () -> ())
```

⟨Mmm.initial_navigator() *set preferences* 207h⟩≡ (33a)

```
  preferences := Mmmprefs.f preffile;
  !preferences();
```

⟨Main.main() *locals* 207i⟩+≡ (29c) ◁188g 208k▷

```
  let preffile = ref (Mmm.user_file "MMM.ad") in
```

⟨`Main.main()` *command line options* 208a⟩+≡ (29c) ◁188h 208h▷
```
  "-prefs", Arg.String (fun s -> preffile := (Fpath.v s)),
  " <file> Preference File";
```

⟨*signature* `Mmm.user_file` 208b⟩≡ (459)
```
  val user_file : string -> Fpath.t
      (* [user_file base] returns $HOME/.mmm/[base] *)
```

⟨*function* `Mmm.user_file` 208c⟩≡ (460)
```
  let user_file (name : string) : Fpath.t =
    home / ".mmm" / name
```

⟨*constant* `Mmm.home` 208d⟩≡ (460)
```
  let home : Fpath.t =
    try
      Fpath.v (Sys.getenv "HOME")
    with Not_found ->
      Logs.err (fun m -> m "Please set the HOME environment variable.");
      raise (Exit.ExitCode (-1))
```

⟨`Main.main()` *user preferences file* 208e⟩≡ (29c)
```
  localize !preffile
```

19.8.2 Initial geometry, `mmm -geometry`

⟨*signature* `Mmm.initial_geom` 208f⟩≡ (459)
```
  val initial_geom : string option ref
```

⟨*constant* `Mmm.initial_geom` 208g⟩≡ (460)
```
  let initial_geom = ref None
```

⟨`Main.main()` *command line options* 208h⟩+≡ (29c) ◁208a 208l▷
```
  "-geometry", Arg.String (fun s -> Mmm.initial_geom := Some s),
  " <wxh+x+y> Initial geometry for the first navigator";
```

⟨`Mmm.navigator()` *set geometry if specified* 208i⟩≡ (41a)
```
  (match !initial_geom with
   | None -> ()
   | Some g -> Wm.geometry_set top g
   );
```

19.9 TK customization

19.9.1 Color palette, `mmm -palette`

⟨`Main.main()` *resource initialisation* 208j⟩+≡ (30c) ◁31d
```
  begin match !palette with
  | None -> ()
  | Some bg -> try Palette.set_background (Tk.NamedColor bg) with _ -> ()
  end;
```

⟨`Main.main()` *locals* 208k⟩+≡ (29c) ◁207i 209b▷
```
  let palette = ref None in
```

⟨`Main.main()` *command line options* 208l⟩+≡ (29c) ◁208h 209c▷
```
  "-palette", Arg.String (fun s -> palette := Some s),
  " <color> Tk Palette";
```

19.9.2 Click to follow, `mmm -clicktofocus`

⟨`Main.main()` *tk initialisation* 209a⟩+≡ (30c) ◁30d

```
  if not !clicktofocus
  then Focus.follows_mouse();
```

⟨`Main.main()` *locals* 209b⟩+≡ (29c) ◁208k 219c▷

```
  let clicktofocus = ref false in
```

⟨`Main.main()` *command line options* 209c⟩+≡ (29c) ◁208l 209f▷

```
  "-clicktofocus", Arg.Unit (fun () -> clicktofocus := true),
  " Click to Focus mode (default is Focus Follows Mouse)";
```

19.10 Help URL, `mmm -helpurl`

⟨*signature* `Mmm.helpurl` 209d⟩≡ (459)

```
  (* Preferences, options *)
  val helpurl : Url.t ref
```

⟨*constant* `Mmm.helpurl` 209e⟩≡ (460)

```
  let helpurl = ref (Lexurl.make (Version.helpurl (Lang.lang ())))
```

⟨`Main.main()` *command line options* 209f⟩+≡ (29c) ◁209c 219d▷

```
  "-helpurl", Arg.String (fun s -> Mmm.helpurl := Lexurl.make s),
  " <url> Help URL";
```

⟨*signature* `Version.helpurl` 209g⟩≡ (285e)

```
  val helpurl : string -> string (* help url *)
```

⟨*function* `Version.helpurl` 209h⟩≡ (286a)

```
  let helpurl = function
    | "iso8859" ->
        Printf.sprintf "http://pauillac.inria.fr/mmm/v%d/docindex.html" number
    | _ -> assert false
```

⟨*Help menu elements* 209i⟩+≡ (47b) ◁47d

```
  Menu.add_command helpm
    [Label (s_ "Help on MMM");
     Command (fun () -> navigator caps false !helpurl |> ignore)];
```

19.11 Extra Protocols

19.11.1 `file://`

⟨*toplevel* `Protos._7` 209j⟩≡ (318a)

```
  let _ = Hashtbl.add protos Url.FILE ((fun _caps -> File.request), Cache.dummy)
```

⟨Lexurl.f *protocol cases* 210a⟩+≡ (51d) ◁51e 213b▷
```
  (* the spec says file://host/ dammit *)
  | "FILE" ->
      (try
         slashslash lexbuf;
         let h = fhost lexbuf in
         let p = slashpath lexbuf in
         result.protocol <- FILE;
         result.host <- h;
         result.path <- p
       with Url_Lexing _ ->
         let p = slashpath lexbuf in
         result.protocol <- FILE;
         result.path <- p
      )
```

⟨*functions* Lexurl.xxx 210b⟩+≡ (288b) ◁54b 216a▷
```
  and fhost = parse
    ['A'-'Z' 'a'-'z' '0'-'9' '.' '-']+
       { Some (normalize_host (Lexing.lexeme lexbuf)) }
  | ""
       { Some "localhost" }
```

⟨*signature* File.request 210c⟩≡ (316a)
```
  (* file: protocol *)
  val request : Www.request -> Document.continuation -> Www.aborter
      (* [request wr cont] returns [abort] *)
```

⟨*exception* File.File_error 210d⟩≡ (316)
```
  exception File_error of string
```

⟨*function* File.document_id 210e⟩≡ (316g)
```
  let document_id wwwr =
    { document_url = wwwr.www_url; document_stamp = no_stamp}
```

⟨*function* File.request 210f⟩≡ (316g)
```
  (*
   * Display a file on the local unix file system (file:)
   *  is path really supposed to be absolute ?
   * Note: completely ignores method (GET, POST,...)
   *)

  let request wr cont =
    let path =
      match wr.www_url.path with
      | Some path -> "/" ^ (Lexurl.remove_dots path)
      | None -> "/"
    in
    ⟨File.request() if CGI path 239b⟩
    else   (* A bit weird, but we don't want to capture errors from the cont *)
      let st =
        try stat path
        with _ -> raise (File_error (s_ "cannot stat file"))
      in
      match st.st_kind with
      | S_REG ->
        begin
        (* check if this is an update *)
          try
            let since = Http_headers.get_header "if-modified-since" wr.www_headers in
            let ht = Lexdate.ht_of_string since in
            let filet = Http_date.ht_of_stamp st.st_mtime in
            if Http_date.compare filet ht > 0
            then raise Not_found (* fall through *)
            else begin
              let dh = {
                document_id = document_id wr;
                document_referer = wr.www_link.h_context;
                document_status = 304;
                dh_headers = [ sprintf "Date: %s" (Date.asc_now())];
                document_feed = (let fd = openfile "/dev/null" [O_RDONLY] 0 in
                  Feed.make_feed fd (Low.count_read (Unix.read fd)));
                document_fragment = wr.www_fragment;
                document_logger = Document.tty_logger
              } in

              Retype.f dh; (* pad: to get the content type based on the suffix *)

              cont.document_process dh;
              (fun () -> ())
            end
         with
         | Not_found  (* default case *)
         | Lexdate.Invalid_date (_,_) ->
            let s =
              try openfile path [O_RDONLY] 0
              with Unix_error(_,_,_) ->
                raise (File_error (s_ "cannot open file"))
            in
            let dh =
              { document_id = document_id wr;
                document_referer = wr.www_link.h_context;
                document_status = 200;
                dh_headers =
                  [sprintf "Content-Length: %d" st.st_size;
                   sprintf "Date: %s" (Date.asc_now());
                   sprintf "Last-modified: %s" (Date.asc st.st_mtime)
                  ];
                document_feed = Feed.make_feed s (Low.count_read (Unix.read s));
                document_fragment = wr.www_fragment;
                document_logger = Document.tty_logger
              } in
             Retype.f dh;
             cont.document_process dh;
             (fun () -> ())
      end
      | S_DIR ->
          let s = dir path in
          cont.document_process
            { document_id = document_id wr;
              document_referer = wr.www_link.h_context;
              document_status = 200;
              dh_headers = ["Content-Type: text/html"];
              document_feed = Feed.make_feed s (Low.count_read (Unix.read s));
              document_fragment = wr.www_fragment;
              document_logger = Document.tty_logger
             };
           (fun () -> ())

      | _ -> raise (File_error (s_ "cannot open file"))
```

Files

Directories

⟨*function* File.dir 212a⟩≡ (316g)
```
  (* It's easiest to do it asynchronously anyway *)
  let dir path =
    try
      let d = opendir path in
      let cin, cout = pipe() in
      match Low.fork() with
      (* child *)
      | 0 ->
          close cin; dup2 cout stdout; close cout;
          begin
            try
              d2html path d
            with e ->
              print_endline (Printexc.to_string e)
          end;
          flush Stdlib.stdout; (* strange bug with our at_exit stuff *)
          (* nosemgrep: do-not-use-exit *)
          exit 0
          (*cin (*duh*) *)
       (* parent *)
       | _n -> closedir d; close cout; cin
    with Unix_error(_,_,_)  ->
      raise (File_error (s_ "cannot open dir"))
```

⟨*function* File.d2html 212b⟩≡ (316g)
```
  let d2html path (d : Unix.dir_handle) =
    (* make sure that when path is used in url, it is / terminated *)
    let pathurl =
      let l = String.length path in
      if l = 0
      then path
      else
        if path.[l-1] = '/'
        then path
        else sprintf "%s/" path
    in
    Printf.printf
  "<HTML>
  <HEAD><TITLE>%s</TITLE>
  <BASE HREF=\"file://localhost%s\">
  </HEAD>
  <BODY>
  <H1>Directory list: %s</H1>
  <PRE>" path pathurl path;

    let entries = ref [] in
    begin
     try
       while true do
         entries := (readdir d) :: !entries
       done
     with  End_of_file -> closedir d
    end;
    entries := List.sort Stdlib.compare !entries;
    !entries |> List.iter (function
      |  "." -> ()
      | ".." ->
         printf "Dir   <A HREF=\"file://localhost%s\">..</A>\n"
                (Filename.concat (dirname (dirname pathurl)) "")
      | f ->
         try
           let fullname = Filename.concat path f in
           let st = stat fullname in
           match st.st_kind with
           | S_DIR -> printf "Dir   <A HREF=\"%s\">%s</A>\n" f f
           | S_REG -> printf "File  <A HREF=\"%s\">%-30s</A>%8d bytes\n"
                         f f (st.st_size)
           | S_LNK -> printf "Link  <A HREF=\"%s\">%s</A>\n" f f
           | _ -> ()
         with Unix_error(_,_,_) -> ()
      );
    printf "</PRE></BODY></HTML>"
```

⟨*function* File.isdir 213a⟩≡ (316g)

```
  (*
   * Simulate directory
   *)
  let _isdir path f =
    let fullname = Filename.concat path f in
    (stat fullname).st_kind = S_DIR
```

19.11.2 mailto://

⟨Lexurl.f *protocol cases* 213b⟩+≡ (51d) ◁210a 215h▷

```
  | "MAILTO" ->
      let address = any lexbuf in
      result.protocol <- MAILTO;
      result.path <- address
```

⟨Nav.request.handle_wr() *match protocol special cases* 213c⟩≡ (37a)

```
  | MAILTO -> Mailto.f wr
   (* mailto: is really a pain. It doesn't fit the retrieval semantics
    * of WWW requests. *)
```

⟨*type* Mailto.msg 213d⟩≡ (317a)

```
  type msg = {
    dest    : string;
    subject : string;
    body    : string
  }
```

⟨*function* Mailto.f 213e⟩≡ (317a)

```
  let f wr =
    match wr.www_url.path with
    | Some raw_address ->
        let address = Urlenc.decode raw_address in
        (match wr.www_link.h_method with
        | GET -> get address wr.www_link.h_context
        | POST d ->
            if wr.www_error#choose
              (s_ "About to send mail with POST data to\n%s" address)
            then
              let subject =
                match wr.www_link.h_context with
                | None -> "no subject"
              | Some s -> "POST data for "^s
            in
            sendmail  { dest = address; subject = subject; body = d}
          else ()
      | _ ->
          wr.www_error#f (s_ "Unsupported method for mailto:")
      )
    | None -> wr.www_error#f (s_ "No address given for mailto:")
```

⟨*constant* Mailto.mailer 214a⟩≡ (317a)
```
  let mailer = ref ""
```

⟨*function* Mailto.get 214b⟩≡ (317a)
```
  let get mailaddr referer =
    let subject =
      match referer with
      | None -> "no subject"
      | Some s -> "About url "^s
    in
    match !mailer with
     | "" -> internal mailaddr subject
     | s ->
        Munix.system_eval s ["_", "-s"; "SUBJECT", subject; "TO", mailaddr] true
         |> ignore
```

⟨*function* Mailto.internal 214c⟩≡ (317a)
```
  let internal address referer =
    !internal_backend address referer
```

⟨*global* Mailto.internal_backend 214d⟩≡ (317a)
```
  let internal_backend =
    ref (fun _ _ -> failwith "no Mailto.internal defined")
```

⟨*function* Mailto.sendmail 214e⟩≡ (317a)
```
  (* if the mail contains a dot line, we're f*cked *)
  let sendmail msg =
    let cmd = try Sys.getenv "MMM_MAIL" with Not_found -> "mail" in
    try
      let (fd_in,fd_out) = pipe() in
      match Low.fork () with
      | 0 ->
          close fd_out;
          dup2 fd_in stdin;
          Munix.execvp cmd [| cmd; "-s"; msg.subject; msg.dest |]
      | n ->
         close fd_in;
         Munix.write_string fd_out msg.body;
         close fd_out;
         (match waitpid [] n with
         | _, WEXITED 0 -> Error.ok (s_ "Mail sent")
         | _, _ -> error msg.body
         )
     with Unix_error(_,_,_) -> error msg.body
```

⟨*function* Mailto.error 214f⟩≡ (317a)
```
  let error body =
    try
      let oc = open_out_bin (Filename.concat (getenv "HOME") "dead.letter") in
      output_string oc body;
      close_out oc;
      Error.f (s_ "Can't send mail (saved in $HOME/dead.letter)")
    with _ ->
      Error.f (s_ "Can't send mail, can't save dead.letter")
```

19.11.3 Proxied protocols

⟨*toplevel* Protos._1 215a⟩≡ (318a)
```
  let _ = Hashtbl.add protos Url.FTP (Http.proxy_req, Cache.tobuffer)
```

⟨*toplevel* Protos._3 215b⟩≡ (318a)
```
  let _ = Hashtbl.add protos Url.GOPHER (Http.proxy_req, Cache.tobuffer)
```

⟨*toplevel* Protos._4 215c⟩≡ (318a)
```
  let _ = Hashtbl.add protos Url.NEWS (Http.proxy_req, Cache.tobuffer)
```

⟨*toplevel* Protos._5 215d⟩≡ (318a)
```
  let _ = Hashtbl.add protos Url.NNTP (Http.proxy_req, Cache.tobuffer)
```

⟨*toplevel* Protos._6 215e⟩≡ (318a)
```
  let _ = Hashtbl.add protos Url.WAIS (Http.proxy_req, Cache.tobuffer)
```

⟨*signature* Http.proxy_req 215f⟩≡ (306c)
```
  val proxy_req: < Cap.network; ..> ->
    Www.request -> Document.continuation -> Www.aborter
```

⟨*function* Http.prox_req 215g⟩≡ (306d)
```
  (* Retrieve.f -> <> (via protos) *)
  let proxy_req caps wr cont =
    let cnx = proxy_request caps wr cont in
    (fun () -> cnx#abort)
```

19.11.4 ftp://

⟨Lexurl.f *protocol cases* 215h⟩+≡ (51d) ◁213b 216b▷
```
  | "FTP" -> (* we don't need the detail of path *)
      slashslash lexbuf;
      let u, p = userpass lexbuf in
      let h, po = hostport lexbuf in
      let path = slashpath lexbuf in
      result.protocol <- FTP;
      result.user <- u;
      result.password <- p;
      result.host <- h;
      result.port <- normalize_port (FTP, po);
      result.path <- path
```

⟨*function* Lexurl.userpass 215i⟩≡ (288b)
```
  and userpass = parse
    (* foo:bar@, foo:@ *)
    [^ ':' '/' '@']+ ':' [^ ':' '/' '@']* '@'
      { let lexeme = Lexing.lexeme lexbuf in
        let pos = String.index lexeme ':' in
        Some (String.sub lexeme 0 pos),
        Some (String.sub lexeme (succ pos) (String.length lexeme - 2 - pos))
      }
    (* foo@, @ *)
  | [^ ':' '/' '@']* '@'
     { let lexeme = Lexing.lexeme lexbuf in
       Some (String.sub lexeme 0 (String.length lexeme - 1)), None
     }

  | ""
     { None, None }
```

⟨*functions* Lexurl.xxx 216a⟩+≡ (288b) ◁210b
```
  and slashpath = parse
    "/" { any lexbuf }
  | ""  { None }
```

19.11.5 telnet://

⟨Lexurl.f *protocol cases* 216b⟩+≡ (51d) ◁215h 216c▷
```
  | "TELNET" ->
      slashslash lexbuf;
      let u,p = userpass lexbuf in
      let h,po = hostport lexbuf in
      result.protocol <- TELNET;
      result.user <- u;
      result.password <- p;
      result.host <- h;
      result.port <- po
```

19.11.6 nntp://

⟨Lexurl.f *protocol cases* 216c⟩+≡ (51d) ◁216b 216d▷
```
  | "NNTP" ->
      let h,po = hostport lexbuf in
      let blah = any lexbuf in
      result.protocol <- NEWS;
      result.host <- h;
      result.port <- po;
      result.path <- blah
```

19.11.7 Old protocols

⟨Lexurl.f *protocol cases* 216d⟩+≡ (51d) ◁216c
```
  | "GOPHER" | "GOPHER+" -> (* we don't need the detail of path *)
      slashslash lexbuf;
      let h,po = hostport lexbuf in
      let path = slashpath lexbuf in
      result.protocol <- GOPHER;
      result.host <- h;
      result.port <- po;
      result.path <- path
  | "NEWS" ->
      let blah = any lexbuf in
      result.protocol <- NEWS;
      result.path <- blah
  | "WAIS" ->
      slashslash lexbuf;
      let h,po = hostport lexbuf in
      let pa,se = pathsearch lexbuf in
      result.protocol <- WAIS;
      result.host <- h;
      result.port <- po;
      result.path <- pa;
      result.search <- se
  | "PROSPERO" ->
      slashslash lexbuf;
      let h,po = hostport lexbuf in
      let p = slashpath lexbuf in
```

```
    result.protocol <- PROSPERO;
    result.host <- h;
    result.port <- po;
    result.path <- p
```

19.12 Gzip decoders

⟨Http_headers.suffixes *elements* 217a⟩+≡ (116c) ◁176c

```
  "gz",  ContentEncoding  "Content-Encoding: gzip";
  "Z",  ContentEncoding  "Content-Encoding: compress";

  "asc", ContentEncoding  "Content-Encoding: pgp";
  "pgp", ContentEncoding  "Content-Encoding: pgp";
```

⟨*signature* Decoders.insert 217b⟩≡ (332)

```
  val insert   : Document.handle -> Document.handle
```

⟨*signature* Decoders.add 217c⟩≡ (332)

```
  val add : string -> (Document.handle -> Document.handle) -> unit
```

⟨*constant* Decoders.decoders 217d⟩≡ (333a)

```
  (* Insert a decoding if necessary.
   * We don't do it in http, since we don't want do decompress when we
   * save for example.
   *)
  let decoders = Hashtbl.create 37
```

⟨*toplevel* Decoders._1 217e⟩≡ (333a)

```
  let _ =
    [ "COMPRESS"   , gzip;
      "X-COMPRESS" , gzip;
      "GZIP"       , gzip;
      "X-GZIP"     , gzip
    ] |> List.iter (fun (s,t) -> Hashtbl.add decoders s t)
```

⟨*function* Decoders.gzip 217f⟩≡ (333a)

```
  (* Note: we must use the feed interface to read from the old dh,
   * and not read directly from the feed_internal file descriptor, because
   * the feed might implement side effects (such as caching).
   * Since we are reading and writing to the same process, we might get
   * deadlocked if we don't watch writes.
   *)
  let gzip dh =
    let (gread, mwrite) = pipe()
    and (mread, gwrite) = pipe()
    in
    Unix.set_close_on_exec mread;
    Unix.set_close_on_exec mwrite;

    match Low.fork() with
      0 ->
        dup2 gread stdin; dup2 gwrite stdout;
        Munix.execvp "gunzip" [| "gunzip"; "-c" |]
        (* dh (* fake *) *)
    | _n ->
        close gread; close gwrite;
       (* it is safe to close feed because the son has a copy *)
        let newdh =
         { dh with
           dh_headers = rem_contentencoding dh.dh_headers;
           document_feed = Feed.make_feed mread (Low.count_read (Unix.read mread));
         }
        in
        let buffer = Bytes.create 4096 in
        let rec copy () =
        try
          let n = dh.document_feed.feed_read buffer 0 4096 in
              if n = 0 then (dclose true dh; close mwrite)
              else begin
            dh.document_feed.feed_unschedule();
            Fileevent.add_fileoutput mwrite
          (fun () ->
            ignore (write mwrite buffer 0 n);
            Fileevent.remove_fileoutput mwrite;
            dh.document_feed.feed_schedule copy)
          end
        with
          Unix_error(_e,_,_) -> dclose true dh; close mwrite
        in
        dh.document_feed.feed_schedule copy;
        newdh
```

⟨*constant* `Decoders.add` 218a⟩≡ (333a)

```
  let add = Hashtbl.add decoders
```

⟨*function* `Decoders.insert` 218b⟩≡ (333a)

```
  let insert dh =
  (* CERN proxy sets Content-Encoding when return code = 500 ! *)
    if dh.document_status >= 400 then dh else
    try
      Hashtbl.find decoders (String.uppercase_ascii (contentencoding dh.dh_headers)) dh
    with
      Not_found -> dh
    | Unix_error(_,_,_) -> dh
```

19.13 Animaged GIFs

⟨*signature* `Img.gif_anim_load` 218c⟩≡ (320a)

```
  val gif_anim_load : bool ref
```

⟨*constant* `Img.gif_anim_load` 218d⟩≡ (320c)

```
  (* Images are a special case of embedded data, because Tk caches them
     internally. Thus, we attempt to maintain our own cache logic above
     Tk's one
   *)

  let gif_anim_load = ref false
```

⟨*signature* `Imgload.gif_anim_auto` 218e⟩≡ (376c)

```
  val gif_anim_auto : bool ref
```

⟨*constant* `Imgload.gif_anim_auto` 218f⟩≡ (380b)

```
  let gif_anim_auto = ref false
```

19.14 Audio

⟨*function* Audio.fake_embed 219a⟩≡ (464b)
```
(* Defines embedded viewer for audio types as re-running the document *)
let fake_embed media_pars w ctx dh =
  Document.dclose true dh;
  try
    let hlink = Hyper.default_link (Url.string_of dh.document_id.document_url)in
    pack [Label.create w [Text "Redispatched externally"]][];
    ctx#goto hlink
  with
    Not_found (* goto *) ->
      pack [Label.create w [Text "No navigation given to us"]][]
  | e ->
      pack [Label.create w [Text (Printexc.to_string e)]][]
```

⟨*toplevel* Audio._1 219b⟩≡ (464b)
```
let _ =
  Mmm.add_embedded_viewer ("audio", "*") fake_embed
```

19.15 video

19.16 mmm_remote

19.16.1 Server

⟨Main.main() *locals* 219c⟩+≡ (29c) ◁209b
```
let accept_external = ref false in
```

⟨Main.main() *command line options* 219d⟩+≡ (29c) ◁209f 221d▷
```
"-external", Arg.Unit (fun () -> accept_external := true),
" Accept remote command (mmm_remote <url>)";
```

⟨Main.main() *mmm server initialisation* 219e⟩≡ (30c)
```
(* This must occur after most initialisations *)
if !accept_external
then Cci.init caps;
```

19.16.2 Client

⟨*function* Main_remote.request 219f⟩≡ (464a)
```
let request sock (cmd : string) (url : string) =
  if cmd <> ""
  then Unix.write sock (Bytes.of_string cmd) 0 (String.length cmd) |> ignore;

  Unix.write sock (Bytes.of_string url) 0 (String.length url) |> ignore;
  Unix.write sock (Bytes.of_string "\n") 0 1 |> ignore;
  let buf = Bytes.create 1024 in
  try
    while true do
      let n = Unix.read sock buf 0 1024 in
      if n = 0
      then raise End_of_file
      else ignore (Unix.write Unix.stdout buf 0 n)
    done
  with End_of_file -> Unix.close sock
```

⟨*function* Main_remote.main 220a⟩≡ (464a)
```
  let main (caps: < caps; Cap.stdout; Cap.stderr; ..>) (argv : string array)
      : Exit.t =
    let file =
      Filename.concat (Filename.concat (Sys.getenv "HOME") ".mmm") "remote" in
    let cmd = ref "" in

    let s = Unix.socket PF_UNIX SOCK_STREAM 0 in
    CapUnix.connect caps s (ADDR_UNIX file);
    Arg_.parse_argv caps argv  [
      "-get", Arg.Unit (fun () -> cmd := "GET "), "Get document";
      "-getbody", Arg.Unit (fun () -> cmd := "GETB "), "Get document body";
      "-head", Arg.Unit (fun () -> cmd := "HEAD "), "Get document headers";
      "-show", Arg.Unit (fun () -> cmd := "DISPLAY "), "Open browser on this URL";
    ]
      (fun url -> request s !cmd url)
      "Usage: mmm_remote [-get | -getbody | -head | -show] <url>\n
       The default is -show.";
     Exit.OK
```

⟨*toplevel* Main_remote._1 220b⟩≡ (464a)
```
  let _ =
    Cap.main (fun (caps : Cap.all_caps) ->
      let argv = CapSys.argv caps in
      Exit.exit caps (Exit.catch (fun () -> main caps argv)))
```

Chapter 20

Advanced Topics

20.1 Proxy

⟨*signature* Http.proxy_xxx 221a⟩≡ (306c)
```
  val proxy: string ref
  val proxy_port: int ref
```

⟨*global* Http.proxy 221b⟩≡ (306d)
```
  (* Default proxy definitions *)
  let proxy = ref "no-proxy-defined"
```

⟨*global* Http.proxy_port 221c⟩≡ (306d)
```
  let proxy_port = ref 80
```

⟨Main.main() *command line options* 221d⟩+≡ (29c) ◁219d 221e▷
```
  "-proxy", Arg.String (fun s -> Http.proxy := s),
  " <hostname> Proxy host";
```

⟨Main.main() *command line options* 221e⟩+≡ (29c) ◁221d 224e▷
```
  "-port", Arg.Int (fun i -> Http.proxy_port := i),
  " <port> Proxy port";
```

⟨*constant* Http.always_proxy 221f⟩≡ (306d)
```
  let always_proxy = ref false
```

⟨Http.request() *if always proxy* 221g⟩≡ (98d)
```
  if !always_proxy
  then proxy_request caps wr cont
```

⟨Http.request() *if http error on* tcp_connect, *try proxy* 221h⟩≡ (98d)
```
  proxy_request caps wr cont
```

⟨*function* Http.proxy_request 221i⟩≡ (306d)
```
  (* Process an HTTP request using the proxy. We pass on the continuation *)
  let proxy_request caps (wr : Www.request) (cont : Document.continuation) =
    tcp_connect caps !proxy !proxy_port wr.www_logging
         (start_request true wr cont)
         (failed_request wr cont.document_finish)
```

⟨Http.full_request() *url value if proxy mode* 221j⟩≡ (102e)
```
  if proxy_mode
  then Url.string_of wwwr.www_url
```

20.2 Forwarding

⟨*Retrieve code behaviour other elements* 222a⟩≡ (92b) 234a ▷
```
  301, forward_permanent;
  302, forward;
  (* 304, update; *)
```

⟨*function* Retrieve.forward_permanent 222b⟩≡ (318e)
```
  (* 301 Moved permanently *)
  let forward_permanent (wr : Www.request) (dh : Document.handle) =
    try
      let newurl = Http_headers.location dh.dh_headers in
      wr.www_error#ok (s_ "Document moved permanently to\n%s" newurl);
      forward wr dh
    with Not_found ->
      Error (s_ "No Location: in forwarding header")
```

⟨*function* Retrieve.forward 222c⟩≡ (318e)
```
  (* 302 Moved temporarily *)
  let forward (wr : Www.request) (dh : Document.handle) =
    try
      let newurl = Http_headers.location dh.dh_headers in
      if (* do we forward automatically ?*)
        match wr.www_link.h_method with
        | GET -> true
        | POST _ ->
           (* Do NOT redirect automatically if method was POST *)
           wr.www_error#choose (s_ "Destination for your POST request has changed\n  from %s\nto %s\nConfirm acti
              (Url.string_of wr.www_url) newurl)
        | _ -> true
      then begin (* consider forwarding as a link *)
         wr.www_logging "Forwarding";
         Retry { wr.www_link with h_uri = newurl; }
      end else
        (* not forwarding a moved POST. We show the document after all,
          since some people (servers ?) use this trick to show the results
          of a POST, despite what the protocol says about this *)
         Ok
    with Not_found ->
      Error (s_ "No Location: in forwarding header")
```

20.3 Encodings

20.3.1 Base 64

Decoding

⟨*signature* Base64.decode 222d⟩≡ (298b)
```
  val decode : string -> string
```

⟨*function* Base64.decode 222e⟩≡ (298c)
```
  let decode s =
    let rpos = ref 0
    and wpos = ref 0
    and len = String.length s in
    let res = Bytes.create (len / 4 * 3) in
      while !rpos < len do
        let v1 = index64.(Char.code s.[!rpos]) in
        let v2 = index64.(Char.code s.[!rpos + 1]) in
        let v3 = index64.(Char.code s.[!rpos + 2]) in
        let v4 = index64.(Char.code s.[!rpos + 3]) in
        (* each char gives 6 bits *)
        let i = (v1 lsl 18) lor (v2 lsl 12) lor (v3 lsl 6) lor v4 in
        Bytes.set res !wpos (Char.chr (i lsr 16));
        Bytes.set res (!wpos+1) (Char.chr ((i lsr 8) land 0xFF));
        Bytes.set res (!wpos+2) (Char.chr (i land 0xFF));
        rpos := !rpos + 4;
        wpos := !wpos + 3
        done;
    let cut =
      if s.[len - 1] = '=' then
        if s.[len - 2] = '=' then 2
        else 1
      else 0
    in
    Bytes.sub_string res 0 (Bytes.length res - cut)
```

⟨*constant* Base64.index64 223a⟩≡ (298c)

```
(* For basic credentials only *)
(* Encoding is [A-Z][a-z][0-9]+/= *)
(* 3 chars = 24 bits = 4 * 6-bit groups -> 4 chars *)

let index64 = Array.make 128 0
```

⟨*toplevel* Base64._1 223b⟩≡ (298c)

```
(* Init the index *)
let _ =
  for i = 0 to 25 do index64.(i + Char.code 'A') <- i done;
  for i = 0 to 25 do index64.(i + Char.code 'a') <- i + 26 done;
  for i = 0 to 9 do  index64.(i + Char.code '0') <- i + 52 done;
  index64.(Char.code '+') <- 62;
  index64.(Char.code '/') <- 63
```

Encoding

⟨*signature* Base64.encode 223c⟩≡ (298b)

```
(* Base64 encoding (ONLY for Basic authentication) *)
val encode : string -> string
```

⟨*function* Base64.encode 223d⟩≡ (298c)

```
(* Encoding *)
let encode s =
  let rpos = ref 0
  and wpos = ref 0 in
  let origlen = String.length s in
  let s,len = match origlen mod 3 with
      0 -> s, origlen
    | 1 -> s ^ "\000\000", origlen + 2
    | 2 -> s ^ "\000", origlen + 1
    | _ -> assert false
  in
  let res = Bytes.create (len / 3 * 4) in
    while !rpos < len do
      let i1 = Char.code s.[!rpos] in
      let i2 = Char.code s.[!rpos+1] in
      let i3 = Char.code s.[!rpos+2] in
      let i = (i1 lsl 16) lor (i2 lsl 8) lor i3 in
```

```
        Bytes.set res (!wpos)  (char64.((i lsr 18) land 0x3f));
        Bytes.set res (!wpos+1) (char64.((i lsr 12) land 0x3f));
        Bytes.set res (!wpos+2) (char64.((i lsr 6) land 0x3f));
        Bytes.set res (!wpos+3) (char64.(i land 0x3f));
        rpos := !rpos + 3;
        wpos := !wpos + 4
        done;
    (* Correct padding *)
    for i = 1 to len - origlen do Bytes.set res (Bytes.length res - i) '=' done;
    Bytes.to_string res
```

⟨*constant* Base64.char64 224a⟩≡ (298c)

```
  let char64 = Array.make 64 'a'
```

⟨*toplevel* Base64._2 224b⟩≡ (298c)

```
  let _ =
    for i = 0 to 25 do char64.(i) <- Char.chr (Char.code 'A' + i) done;
    for i = 0 to 25 do char64.(i+26) <- Char.chr (Char.code 'a' + i) done;
    for i = 0 to 9 do char64.(i+52) <- Char.chr (Char.code '0' + i) done;
    char64.(62) <- '+';
    char64.(63) <- '/'
```

20.4 i18n

⟨*signature* I18n.language 224c⟩≡ (281b)

```
  val language : string ref
```

⟨*constant* I18n.language 224d⟩≡ (281c)

```
  let language = ref ""
```

⟨Main.main() *command line options* 224e⟩+≡ (29c) ◁221e 224h▷

```
  "-lang", Arg.String (fun s -> I18n.language := s),
  " <lang> I18n language";
```

⟨*signature* I18n.message_file 224f⟩≡ (281b)

```
  val message_file : string ref
```

⟨*constant* I18n.message_file 224g⟩≡ (281c)

```
  let message_file = ref ""
```

⟨Main.main() *command line options* 224h⟩+≡ (29c) ◁224e

```
  "-msgfile", Arg.String (fun s -> I18n.message_file := s),
  " <file> I18n message file";
```

⟨*signature* Lang.lang 224i⟩≡ (274a)

```
  val lang : unit -> string
```

⟨*function* Lang.lang 224j⟩≡ (274b)

```
  let lang () =
    "iso8859"
```

⟨*signature* I18n.sprintf 224k⟩≡ (281b)

```
  val sprintf: ('a, unit, string) format -> 'a
```

⟨*signature* I18n.menu_option 224l⟩≡ (281b)

⟨*signature* I18n.menu_pattern 224m⟩≡ (281b)

⟨*function* I18n.fprintf 225a⟩≡ (281c)
```
  (* Internationalization (translation of error messages) *)

  let fprintf x =
    Printf.fprintf x
```

⟨*function* I18n.sprintf 225b⟩≡ (281c)
```
  let sprintf x =
    Printf.sprintf x
```

⟨*function* I18n.read_transl_file 225c⟩≡ (281c)
```
  let read_transl_file msgfile =
    let ic = open_in msgfile in
    let tag_buffer = Bytes.create 16
    and msg_buffer = Bytes.create 1024 in
    let rec store_tag c i =
      if i >= 16 then i else (Bytes.set tag_buffer i c; succ i)
    and store_msg c i =
      if i >= 1024 then i else (Bytes.set msg_buffer i c; succ i)
    and read_line i =
      match input_char ic with
        '\n' -> i
      | '\\' -> begin match input_char ic with
                  '\\' -> read_line(store_msg '\\' i)
                | 'n'  -> read_line(store_msg '\n' i)
                | '\n' -> skip_blanks i
                | c    -> read_line(store_msg c (store_msg '\\' i))
                end
      | c    -> read_line(store_msg c i)
    and skip_blanks i =
      match input_char ic with
        ' '|'\t' -> skip_blanks i
      | c        -> read_line(store_msg c i)
    and read_tag i =
      match input_char ic with
        ':'           -> (i, skip_blanks 0)
      | ' '|'\n'|'\t' -> read_tag i
      | c             -> read_tag(store_tag c i) in
    let transl_tbl = Hashtbl.create 37 in
    let currsrc = ref "" in
    begin try
      while true do
        let (tag_len, msg_len) = read_tag 0 in
        if Bytes.sub_string tag_buffer 0 tag_len = "src" then
          currsrc := Bytes.sub_string msg_buffer 0 msg_len
        else if Bytes.sub_string tag_buffer 0 tag_len = !language then
          Hashtbl.add transl_tbl !currsrc (Bytes.sub_string msg_buffer 0 msg_len)
        else ()
      done
    with End_of_file ->
      close_in ic
    end;
    transl_tbl
```

⟨*type* I18n.translation_table 225d⟩≡ (281c)
```
  type translation_table =
      Unknown
    | NoTranslation
    | Transl of (string, string) Hashtbl.t
```

⟨*constant* I18n.transl_table 226a⟩≡ (281c)
```
  let transl_table = ref Unknown
```

⟨*function* I18n.translate 226b⟩≡ (281c)
```
  let rec translate msg =
    match !transl_table with
      NoTranslation ->
        msg
    | Transl tbl ->
        begin try Hashtbl.find tbl msg with Not_found -> msg end
    | Unknown ->
        transl_table :=
          if String.length !language == 0 then
            NoTranslation
          else begin
            try
              if Sys.file_exists !message_file then
                Transl(read_transl_file !message_file)
              else NoTranslation
            with Sys_error _ | Sys.Break ->
              NoTranslation
          end;
        translate msg
```

⟨function I18n.fprintf (./commons/i18n.ml) 226c⟩≡ (281c)
```
  let _fprintf oc (fmt : ('a, out_channel, unit) format) =
    fprintf oc
      (Obj.magic(translate(Obj.magic fmt : string)) :
                                   ('a, out_channel, unit) format)
```

⟨function I18n.sprintf (./commons/i18n.ml) 226d⟩≡ (281c)
```
  let sprintf (fmt : ('a, unit, string) format) =
    sprintf
      (Obj.magic(translate(Obj.magic fmt : string)) :
                                   ('a, unit, string) format)
```

⟨*function* I18n.menu_option 226e⟩≡ (281c)

⟨*exception* I18n.Found 226f⟩≡ (281c)

⟨*function* I18n.menu_pattern 226g⟩≡ (281c)

⟨Htmlw.display_html *i18 encoder for forms* 226h⟩≡ (124a)
```
  (* I18n encoder for Forms *)
  (*
  if !Lang.japan then begin
    mach#set_i18n_encoder (fun s -> Charset.encoder feed_read#get_code s)
  end;
  *)
```

⟨Htmlw.display_html *in feed,* End_of_file *exn handler, if japan* 226i⟩≡

⟨Htmlw.display_html *in feed,* End_of_file *exn handler, if japn and charset* 226j⟩≡

⟨Html_disp.machine *i18 methods* 226k⟩≡ (27b)
```
  (* For other languages) *)
  (* encode the internal i18n strings to corresponding encodings *)
  method virtual i18n_encoder : string -> string
  method virtual set_i18n_encoder : (string -> string) -> unit
```

⟨Html_disp.display_machine *i18n methods* 227a⟩≡ (125c)

```
  val mutable i18n_encoder = (fun s -> s : string -> string)
  method i18n_encoder = i18n_encoder
  method set_i18n_encoder enc = i18n_encoder <- enc
```

20.4.1 Special characters

⟨latin1_normal *elements* 227b⟩≡ (67g)

```
  (* old: we were generating single-byte latin1 encoding of the special
   * symbols but better to generated instead UTF8 chars as Tk assumes UTF8
   *)
  "nbsp",  "\194\160"; (* U+00A0 non-breaking space *)
  "iexcl", "\194\161"; (* U+00A1 inverted exclamation mark *)
  "cent",  "\194\162"; (* U+00A2 cent sign*)
  "pound", "\194\163"; (* U+00A3 pound sterling sign*)
  "curren", "\194\164"; (* U+00A4 general currency sign*)
  "yen", "\194\165"; (* U+00A5 yen sign*)
  "brvbar", "\194\166"; (* U+00A6 broken (vertical) bar *)
  "sect", "\194\167"; (* U+00A7 section sign *)
  "uml", "\194\168"; (* U+00A8 umlaut (dieresis) *)
  "copy", "\194\169"; (* U+00A9 copyright sign *)
  "ordf", "\194\170"; (* U+00AA ordinal indicator, feminine *)
  "laquo", "\194\171"; (* U+00AB angle quotation mark, left *)
  "not", "\194\172"; (* U+00AC not sign *)
  "shy", "\194\173"; (* U+00AD soft hyphen *)
  "reg", "\194\174"; (* U+00AE registered sign *)
  "macr", "\194\175"; (* U+00AF macron *)
  "deg", "\194\176"; (* U+00B0 degree sign *)
  "plusmn", "\194\177"; (* U+00B1 plus-or-minus sign *)
  "sup2", "\194\178"; (* U+00B2 superscript two *)
  "sup3", "\194\179"; (* U+00B3 superscript three *)
  "acute", "\194\180"; (* U+00B4 acute accent *)
  "micro", "\194\181"; (* U+00B5 micro sign *)
  "para", "\194\182"; (* U+00B6 pilcrow (paragraph sign) *)
  "middot", "\194\183"; (* U+00B7 middle dot *)
  "cedil", "\194\184"; (* U+00B8 cedilla *)
  "sup1", "\194\185"; (* U+00B9 superscript one *)
  "ordm", "\194\186"; (* U+00BA ordinal indicator, masculine *)
  "raquo", "\194\187"; (* U+00BB angle quotation mark, right *)
  "frac14", "\194\188"; (* U+00BC fraction one-quarter *)
  "frac12", "\194\189"; (* U+00BD fraction one-half *)
  "frac34", "\194\190"; (* U+00BE fraction three-quarters *)
  "iquest", "\194\191"; (* U+00BF inverted question mark *)
  "Agrave",  "\195\128"; (* U+00C0 capital A, grave accent *)
  "Aacute",  "\195\129"; (* U+00C1 capital A, acute accent *)
  "Acirc",  "\195\130"; (* U+00C2 capital A, circumflex accent *)
  "Atilde",  "\195\131"; (* U+00C3 capital A, tilde *)
  "Auml",  "\195\132"; (* U+00C4 capital A, dieresis or umlaut mark *)
  "Aring",  "\195\133"; (* U+00C5 capital A, ring *)
  "AElig",  "\195\134"; (* U+00C6 capital AE diphthong (ligature) *)
  "Ccedil",  "\195\135"; (* U+00C7 capital C, cedilla *)
  "Egrave",  "\195\136"; (* U+00C8 capital E, grave accent *)
  "Eacute",  "\195\137"; (* U+00C9 capital E, acute accent *)
  "Ecirc",  "\195\138"; (* U+00CA capital E, circumflex accent *)
  "Euml",  "\195\139"; (* U+00CB capital E, dieresis or umlaut mark *)
  "Igrave",  "\195\140"; (* U+00CC capital I, grave accent *)
  "Iacute",  "\195\141"; (* U+00CD capital I, acute accent *)
  "Icirc",  "\195\142"; (* U+00CE capital I, circumflex accent *)
  "Iuml",  "\195\143"; (* U+00CF capital I, dieresis or umlaut mark *)
  "ETH",  "\195\144"; (* U+00D0 capital Eth, Icelandic *)
  "Ntilde",  "\195\145"; (* U+00D1 capital N, tilde *)
  "Ograve",  "\195\146"; (* U+00D2 capital O, grave accent *)
  "Oacute",  "\195\147"; (* U+00D3 capital O, acute accent *)
  "Ocirc",  "\195\148"; (* U+00D4 capital O, circumflex accent *)
  "Otilde",  "\195\149"; (* U+00D5 capital O, tilde *)
  "Ouml",  "\195\150"; (* U+00D6 capital O, dieresis or umlaut mark *)
  "times", "\195\151"; (* U+00D7 multiply sign*)
  "Oslash",  "\195\152"; (* U+00D8 capital O, slash *)
  "Ugrave",  "\195\153"; (* U+00D9 capital U, grave accent *)
  "Uacute",  "\195\154"; (* U+00DA capital U, acute accent *)
  "Ucirc",  "\195\155"; (* U+00DB capital U, circumflex accent *)
  "Uuml",  "\195\156"; (* U+00DC capital U, dieresis or umlaut mark *)
  "Yacute", "\195\157"; (* U+00DD capital Y, acute accent *)
  "THORN",  "\195\158"; (* U+00DE capital THORN, Icelandic *)
  "szlig",  "\195\159"; (* U+00DF small sharp s, German (sz ligature) *)
  "agrave",  "\195\160"; (* U+00E0 small a, grave accent *)
  "aacute",  "\195\161"; (* U+00E1 small a, acute accent *)
  "acirc",  "\195\162"; (* U+00E2 small a, circumflex accent *)
  "atilde",  "\195\163"; (* U+00E3 small a, tilde *)
  "auml",  "\195\164"; (* U+00E4 small a, dieresis or umlaut mark *)
  "aring",  "\195\165"; (* U+00E5 small a, ring *)
  "aelig",  "\195\166"; (* U+00E6 small ae diphthong (ligature) *)
  "ccedil",  "\195\167"; (* U+00E7 small c, cedilla *)
  "egrave",  "\195\168"; (* U+00E8 small e, grave accent *)
  "eacute",  "\195\169"; (* U+00E9 small e, acute accent *)
  "ecirc",  "\195\170"; (* U+00EA small e, circumflex accent *)
  "euml",  "\195\171"; (* U+00EB small e, dieresis or umlaut mark *)
  "igrave",  "\195\172"; (* U+00EC small i, grave accent *)
  "iacute",  "\195\173"; (* U+00ED small i, acute accent *)
  "icirc",  "\195\174"; (* U+00EE small i, circumflex accent *)
  "iuml",  "\195\175"; (* U+00EF small i, dieresis or umlaut mark *)
  "eth",  "\195\176"; (* U+00F0 small th, Icelandic *)
  "ntilde",  "\195\177"; (* U+00F1 small n, tilde *)
  "ograve",  "\195\178"; (* U+00F2 small o, grave accent *)
  "oacute",  "\195\179"; (* U+00F3 small o, acute accent *)
  "ocirc",  "\195\180"; (* U+00F4 small o, circumflex accent *)
  "otilde",  "\195\181"; (* U+00F5 small o, tilde *)
  "ouml",  "\195\182"; (* U+00F6 small o, dieresis or umlaut mark *)
  "divide", "\195\183"; (* U+00F7 divide sign *)
  "oslash",  "\195\184"; (* U+00F8 small o, slash *)
  "ugrave",  "\195\185"; (* U+00F9 small u, grave accent *)
  "uacute",  "\195\186"; (* U+00FA small u, acute accent *)
  "ucirc",  "\195\187"; (* U+00FB small u, circumflex accent *)
  "uuml",  "\195\188"; (* U+00FC small u, dieresis or umlaut mark *)
  "yacute",  "\195\189"; (* U+00FD small y, acute accent *)
  "thorn",  "\195\190"; (* U+00FE small thorn, Icelandic *)
  "yuml",  "\195\191"  (* U+00FF small y, dieresis or umlaut mark *)
```

20.5 Referer

⟨Http.full_request() *helper functions* 228⟩+≡ (102e) ◁103g 229g▷

```
  let write_referer =
    match wwwr.www_link.h_context with
    | None ->   (fun () -> ())
    | Some r -> (fun () -> if !send_referer then w ("Referer: " ^ r ^ "\r\n"))
  in
```

⟨*constant* Http.send_referer 229a⟩≡ (306d)
```
(*
 * HTTP/1.0
 * Headers should be configurable
 *)

let send_referer = ref false
```

⟨Document.handle *other fields* 229b⟩+≡ (22b) ◁199a 244e▷
```
document_referer: string option;
  (* URL of refering document, if any *)
```

20.6 Security

⟨Www.request *security field* 229c⟩≡ (18f)
```
mutable www_auth : (string * string) list;  (* basic auth *)
```

⟨Messages.request_message *other fields* 229d⟩≡ (22c)
```
request_auth : (string * string) option;
         (* have we authentified the emitter (authtype, authuser) *)
```

⟨*Other menu elements* 229e⟩≡ (47a)
```
Menu.add_command othersm
  [Label (s_ "Load Authorizations..."); Command Auth.load];
Menu.add_command othersm
  [Label (s_ "Edit Authorizations..."); Command Auth.edit];
Menu.add_command othersm
  [Label (s_ "Save Authorizations..."); Command Auth.save];
```

⟨Http.full_request() *write auth stuff* 229f⟩≡ (104 103c)
```
write_realm_auth ();
if proxy_mode
then write_proxy_auth();
```

⟨Http.full_request() *helper functions* 229g⟩+≡ (102e) ◁228 229h▷
```
(* If the request has an Authorization, write it *)
let write_realm_auth () =
  try
    let cookie = List.assoc "realm" wwwr.www_auth in
    w ("Authorization: Basic "^cookie^"\r\n")
  with Not_found -> ()
in
```

⟨Http.full_request() *helper functions* 229h⟩+≡ (102e) ◁229g
```
(* For proxy, we don't wait until we get an authorization error *)
let write_proxy_auth () =
  let authspace = Auth.{
    auth_proxy = true;
    auth_host = !proxy;
    auth_port = !proxy_port;
    auth_dir = "";
    auth_realm = ""}
  in
  try (* do we know the cookie *)
    let cookie = Auth.get authspace in
    w ("Proxy-Authorization: Basic "^cookie^"\r\n")
  with Not_found -> (* is that in the request ? *)
     try
```

```
        let cookie = List.assoc "proxy" wwwr.www_auth in
        w ("Proxy-Authorization: Basic "^cookie^"\r\n")
      with Not_found -> ()
  in
```

20.6.1 Parsing

⟨*signature* Lexheaders.challenge 230a⟩≡ (304a)

```
  val challenge : Lexing.lexbuf -> Http_headers.authChallenge
```

20.6.2 Security challenge

⟨*signature* Http_headers.challenge 230b⟩≡ (300g)

```
  val challenge : Messages.header list -> string
    (* WWW-Authenticate *)
```

⟨*signature* Http_headers.proxy_challenge 230c⟩≡ (300g)

```
  val proxy_challenge : Messages.header list -> string
    (* Proxy-Authenticate *)
```

⟨*signature* Http_headers.expires 230d⟩≡ (300g)

```
  val expires : Messages.header list -> Http_date.http_time option
    (* Expires *)
```

⟨*type* Http_headers.authScheme 230e⟩≡ (302 300g)

```
  (* Authorisation headers *)
  type authScheme =
      AuthBasic
    | AuthExtend of string
```

⟨*type* Http_headers.authChallenge 230f⟩≡ (302 300g)

```
  type authChallenge =
      { challenge_scheme : authScheme;
        challenge_realm : string;
        challenge_params: (string * string) list
      }
```

⟨*type* Auth.authSpace 230g⟩≡ (303)

```
  (* Authorizations are remembered on the base of the directory url and realm
   * They are kept during the whole MMM session, with expiration
   *)
  type authSpace = {
     auth_proxy: bool;
     auth_host : string;
     auth_port : int;
     auth_dir : string;
     auth_realm : string
    }
```

⟨*signature* Auth.lifetime 230h⟩≡ (303a)

```
  val lifetime : int ref
```

⟨*signature* Auth.auth_file 230i⟩≡ (303a)

```
  val auth_file : string ref
```

⟨*signature* Auth.edit 230j⟩≡ (303a)

```
  val edit : unit -> unit
```

⟨*signature* `Auth.load` 231a⟩≡ (303a)
```
  val load : unit -> unit
```

⟨*signature* `Auth.save` 231b⟩≡ (303a)
```
  val save : unit -> unit
```

⟨*signature* `Auth.add` 231c⟩≡ (303a)
```
  val add : authSpace -> string -> unit
```

⟨*signature* `Auth.get` 231d⟩≡ (303a)
```
  val get : authSpace -> string
```

⟨*signature* `Auth.check` 231e⟩≡ (303a)
```
  val check : Www.request -> Http_headers.authChallenge -> authSpace ->
                  (string * bool * authSpace) option
```

⟨*type* `Auth.authEntry` 231f⟩≡ (303b)
```
  type authEntry = {
     auth_cookie : string;
     mutable auth_lastused : float
     }
```

⟨*constant* `Auth.authorizations` 231g⟩≡ (303b)
```
  let authorizations = Hashtbl.create 37
```

⟨*function* `Auth.get` 231h⟩≡ (303b)
```
  let get space =
    let entry = Hashtbl.find authorizations space in
      entry.auth_lastused <- Unix.time();
      entry.auth_cookie
```

⟨*constant* `Auth.lifetime` 231i⟩≡ (303b)
```
  (* Lifetime, in minutes. Default is one hour *)
  let lifetime = ref 60
```

⟨*function* `Auth.lookup` 231j⟩≡ (303b)
```
  let rec lookup space =
    (* Printf.eprintf "%s\n" space.auth_dir; flush Pervasives.stderr; *)
    try
      Hashtbl.find authorizations space
    with
      Not_found ->
       if space.auth_dir = "/" || space.auth_dir = "."
       then raise Not_found
       else
        let newdir = Filename.dirname space.auth_dir in
         lookup {auth_proxy = space.auth_proxy;
                 auth_host = space.auth_host;
              auth_port = space.auth_port;
          auth_dir = newdir;
          auth_realm = space.auth_realm}
```

⟨*function* `Auth.ask_cookie` 231k⟩≡ (303b)
```
  let ask_cookie forwhere =
    try
      let u,p = !open_passwd_ref forwhere in
      Base64.encode (u^":"^p)
    with
    | Failure "cancelled" ->
        failwith "cancelled"
    | _ ->
        Error.f (s_ "Error in base 64 encoding");
        failwith "cancelled"
```

⟨*function* Auth.replace 232a⟩≡ (303b)
```
  let replace kind cookie l =
    let rec repl acc = function
      [] -> (kind,cookie)::acc
    | (k,_)::l when k = kind -> repl (acc) l
    | p::l -> repl (p::acc) l in
    repl [] l
```

⟨*function* Auth.add 232b⟩≡ (303b)
```
  let add space cookie =
    Log.debug "adding cookie";
    Hashtbl.add authorizations
        space
        {auth_cookie = cookie; auth_lastused = Unix.time()}
```

⟨*function* Auth.check 232c⟩≡ (303b)
```
  (* Kind is either: realm or proxy *)
  let check wwwr challenge authspace =
    let kind = if authspace.auth_proxy then "proxy" else "realm" in
    match challenge.challenge_scheme with
      AuthExtend _ -> (* we don't know how to do this *)
         None
    | AuthBasic -> (* params are gleefully ignored *)
       try (* if the passwd request is cancelled *)
        let cookie, isnew =
          if List.mem_assoc kind wwwr.www_auth then begin
             (* we already tried, so the authorization is bad ! *)
             Hashtbl.remove authorizations authspace; (* in case *)
             ask_cookie (s_ "Authorization for %s \"%s\" on \
                                          %s:%d/%s"
                                 kind challenge.challenge_realm
                                 authspace.auth_host authspace.auth_port
                                 authspace.auth_dir),
             true
             end
         else (* ah, it is our first try,  get the authorization *)
           if authspace.auth_proxy then
              ask_cookie (s_ "Authorization for %s \"%s\" on \
                                          %s:%d/%s"
                                 kind challenge.challenge_realm
                                 authspace.auth_host authspace.auth_port
                                 authspace.auth_dir),
              true
           else
             try
               let entry = lookup authspace in
                entry.auth_lastused <- Unix.time();
                entry.auth_cookie, false
             with Not_found ->
              ask_cookie (s_ "Authorization for %s \"%s\" on \
                                          %s:%d/%s"
                                 kind challenge.challenge_realm
                                 authspace.auth_host authspace.auth_port
                                 authspace.auth_dir),
              true
        in
        wwwr.www_auth <- replace kind cookie wwwr.www_auth;
        Some (cookie, isnew, authspace)
       with
        Failure "cancelled" -> None
```

⟨*function* Auth.edit 233a⟩≡ (303b)
```
  (* needs to be refined *)
  let edit () =
    !edit_backend ()
```

⟨*constant* Auth.auth_file 233b⟩≡ (303b)
```
  (* Saving authorizations to file *)
  let auth_file = ref ""
```

⟨*function* Auth.save 233c⟩≡ (303b)
```
  let save () =
   if !auth_file <> "" then
    let auth_file = Msys.tilde_subst !auth_file in
    try
      let o = openfile auth_file [O_WRONLY; O_CREAT] 0o600 in
      let oc = out_channel_of_descr o in
        output_value oc authorizations;
        flush oc;
        close o
    with
    | Unix_error(e,_,_) ->
        Error.f (s_ "Error in authorisation save\n%s" (Unix.error_message e))
    | Sys_error s ->
        Error.f (s_ "Error in authorisation save\n%s" s)
   else
     Error.f (s_ "No authorisation file defined")
```

⟨*function* Auth.load 233d⟩≡ (303b)
```
  let load () =
    if !auth_file <> "" then
      let auth_file = Msys.tilde_subst !auth_file in
      try
        let ic = open_in auth_file in
        let table = input_value ic
        and time = Unix.time() in
         Hashtbl.iter
            (fun spacerealm entry ->
             entry.auth_lastused <- time;
             Hashtbl.add authorizations spacerealm entry)
            table;
      close_in ic
      with Sys_error s ->
        Error.f (s_ "Error in authorisation load\n%s" s)
   else
     Error.f (s_ "No authorisation file defined")
```

⟨*signature* Auth.init 233e⟩≡ (303a)
```
  val init : unit -> unit
```

⟨*function* Auth.init 233f⟩≡ (303b)
```
  let init () =
    let check () =
      let remove = ref []
      and lifetime = float (60 * !lifetime)
      and time = Unix.time () in
      Hashtbl.iter
        (fun space entry ->
         let expiration_time = entry.auth_lastused +. lifetime in
      if time > expiration_time then remove := space :: !remove)
        authorizations;
      List.iter (Hashtbl.remove authorizations) !remove
    in
    let rec tim () =
      Timer_.set (!lifetime * 30000) (fun () -> check(); tim ())
    in
    tim ()
```

20.6.3 HTTP status

⟨*Retrieve code behaviour other elements* 234a⟩+≡ (92b) ◁222a

```
  401, unauthorized;
  407, proxy_unauthorized;
```

⟨*function* Retrieve.ask_auth 234b⟩≡ (318e)

```
  (* 401 Unauthorized *)
  let ask_auth (wr : Www.request) (dh : Document.handle) =
    wr.www_logging (s_ "Checking authentication");
    let rawchallenge = Http_headers.challenge dh.dh_headers in
    let challenge =
      Lexheaders.challenge (Lexing.from_string rawchallenge) in
    let host = match wr.www_url.host with
       Some h -> h
     | None -> ""
    and dir = match wr.www_url.path with
       Some "" -> "/"
     | Some h -> Filename.dirname h
     | None -> "/"
    and port = match wr.www_url.port with
       Some p -> p
     | None -> 80 (* should never happen *) in

    Auth.check wr challenge
        {auth_proxy = false;
         auth_host = host;
         auth_port = port;
         auth_dir = dir;
         auth_realm = challenge.challenge_realm}
```

⟨*function* Retrieve.unauthorized 234c⟩≡ (318e)

```
  let unauthorized wr dh =
    match ask_auth wr dh with
      None -> (* no attempt to answer challenge, display the message *)
        Ok
    | Some (cookie, isnew, space) ->
       (* restart the request with a continuation that says first
      to check if authorization was valid, and then proceed
      to the normal intended continuation *)
       Restart (fun newdh ->
          if newdh.document_status <> 401 && isnew then
            Auth.add space cookie;
          (* Put the challenge header again *)
          begin try
            newdh.dh_headers <- ("WWW-Authenticate: "^ (Http_headers.challenge dh.dh_headers))
                                :: newdh.dh_headers
           with
            Not_found -> ()
          end;
                newdh)
```

⟨*function* `Retrieve.ask_proxy_auth` 235a⟩≡ (318e)

```
  (* 407 Unauthorized *)
  (* We dump the realm altogether, because it has no meaning for proxies *)
  let ask_proxy_auth (wr : Www.request) (dh : Document.handle) =
    wr.www_logging (s_ "Checking proxy authentication");
    let rawchallenge = Http_headers.proxy_challenge dh.dh_headers in
    let challenge =
      Lexheaders.challenge (Lexing.from_string rawchallenge) in
    Auth.check wr challenge
        {auth_proxy = true;
         auth_host = !Http.proxy;
         auth_port = !Http.proxy_port;
         auth_dir = "";
         auth_realm = ""}
```

⟨*function* `Retrieve.proxy_unauthorized` 235b⟩≡ (318e)

```
  let proxy_unauthorized wr dh =
    Log.debug "proxy_unauthorized handler";
    match ask_proxy_auth wr dh with
      None -> (* no attempt to answer challenge, display the message *)
        Ok
    | Some (cookie, isnew, space) ->
       (* restart the request with a continuation that says first
      to check if authorization was valid, and then proceed
      to the normal intended continuation *)
        Restart (fun newdh ->
           Log.debug "proxy_unauthorized wrapper";
           if newdh.document_status <> 407 && isnew then
             Auth.add space cookie;
           (* Put the challenge header again *)
           begin try
             newdh.dh_headers <-
                ("Proxy-Authenticate: "^ (Http_headers.proxy_challenge dh.dh_headers))
                :: newdh.dh_headers
            with
             Not_found -> ()
           end;
                   newdh)
```

20.7 Optimisations

20.7.1 Document cache, `Cache`

⟨*signature* `Cache.tobuffer` 235c⟩≡ (309l)

```
  val tobuffer: Document.handle -> Document.data * cache_fill
```

⟨*function* `Cache.tobuffer` 235d⟩≡ (314e)

```
  let tobuffer _dh =
    let b = Ebuffer.create 1024 in
    MemoryData b, {cache_write = Ebuffer.output b;
                   cache_close = (fun () -> ())}
```

⟨*function* `Cache.add` 235e⟩≡ (314e)

```
  (* Add a new entry *)
  let add did doc =
    if !debug
    then Log.f (sprintf "Adding new cache entry %s(%d) %s"
                      (Url.string_of did.document_url)
```

```
                        did.document_stamp
                        (match doc.document_data with
                        | MemoryData _ -> "in memory"
                        | FileData (f,true) -> !!f
                        | FileData (f,false) -> "fake " ^ !!f)
                        );

    (* Kill the previous entry, if any [for update] *)
    kill did;
    (* Because of frames (not kept in history), we must make room even
     * in history mode
     *)
    if (*not !history_mode && *)!current >= !max_documents
    then make_room()
    else begin
      if !debug
      then Log.f (sprintf "Cache size(max): %d(%d)" !current !max_documents);
      incr current;
      memory := (did,
                 { cache_document = doc;
                   cache_pending = true;
                   cache_lastused = max_lastused;
                   cache_condition = Condition.create()
                 }) :: !memory
    end
```

⟨Nav.request.handle_wr() *if use cache* 236a⟩≡ (37a)

```
  (* If the the document can be cached, then it is with no_stamp *)
  let did = Document.{ document_url = wr.www_url; document_stamp = no_stamp } in
  try
    specific nav did wr
  with Not_found ->
    try
      let doc = Cache.find did in
      try (* display it from source *)
        process nav wr (Cache.make_handle wr doc)
      with
      | Sys_error s ->
          wr.www_error#f (s_
           "Cache error occurred during save of temporary buffer (%s)"
              s)
      | Unix.Unix_error (e,fname,arg) ->
          wr.www_error#f
            (s_ "Cache error occurred when opening temporary file\n%s: %s (%s)"
                   fname (Unix.error_message e) arg)
   with Not_found -> (* we don't have the document *)
     retrieve_and_handle wr
```

⟨*signature* Nav.dont_check_cache 236b⟩≡ (450f)

```
  (*-*)
  val dont_check_cache : Www.request -> bool
```

⟨*function* Nav.dont_check_cache 236c⟩≡ (453)

```
  (* Some requests should not be looked for in the cache *)
  let dont_check_cache (wwwr : Www.request) =
    match wwwr.www_link.h_method with
    | POST _ -> true
    | _ -> false
```

⟨Mmm.navigator() *call* touch_current *to not swap displayed documents* 237a⟩≡ (41a)
```
  (* Yet another timer to avoid flushing displayed documents *)
  let rec touch_current () =
    if Winfo.exists top then begin
      Cache.touch hist.h_current.h_did;
      Timer.set 10000 touch_current;
    end
  in
  touch_current();
```

20.7.2 Graphic cache, Gcache

⟨Mmm.navigator.show_current() *start hook* 237b⟩≡ (38a)
```
  di#di_touch;
```

⟨Viewers.display_info *graphic cache methods signatures* 237c⟩≡ (25a)
```
  method di_touch : unit
  method di_last_used : int
```

⟨Viewers.display_info *graphic cache methods* 237d⟩≡ (340c)
```
  val mutable di_last_used = !Low.global_time
  method di_last_used =
    di_last_used
  method di_touch =
    di_last_used <- !Low.global_time
```

⟨*signature* Viewers.di_compare 237e⟩≡ (339c)
```
  val di_compare : display_info -> display_info -> int
```

⟨*function* Viewers.di_compare 237f⟩≡ (340c)
```
  let di_compare di di' =
    (*di#di_last_used > di'#di_last_used*)
    Stdlib.compare di'#di_last_used di#di_last_used
```

⟨Nav.process_viewer() *add in cache and history the document* 237g⟩+≡ (37c) ◁42e
```
  Gcache.add nav.nav_id dh.document_id di;
```

⟨Mmm.navigator() *destroy navigator hook* 237h⟩+≡ (42b) ◁206i
```
  Gcache.kill hist.h_key;
```

⟨*function* Gcache.remove 237i⟩≡ (434a)
```
  (* Removes a given dinfo for a cached document
   *   used when adding in the middle of the history
   *)
  let remove hkey did =
    if !debug
    then Log.f (sprintf "Removing %s in window %d"
                  (Url.string_of did.document_url) hkey);
    try
      let r = Hashtbl.find table hkey in
      let di = List.assoc did !r in
      di#di_abort;
      di#di_destroy;
      r := Mlist.except_assoc did !r;
      if !Cache.history_mode
      then nocache did
    with Not_found ->
      Log.debug "Gcache.remove failed !"
```

⟨Viewers.display_info *graphic cache virtual methods signatures* 238a⟩≡ (25b)
```
  method virtual di_destroy : unit  (* die *)
```

⟨Plain.plain *graphic cache destroy methods* 238b⟩≡ (113d)
```
  method di_destroy =
    if Winfo.exists frame
    then Tk.destroy frame
```

⟨Htmlw.display_html *graphic cache destroy methods* 238c⟩≡ (122c)
```
  method di_destroy =
    if Winfo.exists frame
    then Tk.destroy frame;
```

⟨Nav.t *graphic cache related methods* 238d⟩≡ (35b)
```
  nav_id : int;  (* key for the gfx cache *)
```

⟨Mmm.navigator() *set nav fields* 238e⟩+≡ (34a) ◁207a
```
  nav_id = hist.h_key;
```

20.8 Frames

⟨*signature* Dtd.dtd32f 238f⟩≡ (292)
```
  val dtd32f : t
```

⟨*constant* Dtd.dtd32f 238g⟩≡ (293a)
```
  (* Add frames somwhere to dtd32.
   * Luckily we chose sets, and they are functional
   *)
  let dtd32f =
    let dtd = {
      dtd_name = "HTML 3.2 + frames";
      contents = Hashtbl.create 101;
      open_omitted = dtd32.open_omitted;
      close_omitted = dtd32.close_omitted;
    } in
    let _omit_open el =
      dtd.open_omitted <- Elements.add el dtd.open_omitted in
    let omit_close el =
      dtd.close_omitted <- Elements.add el dtd.close_omitted in
    let add_elem =
     Hashtbl.add dtd.contents in

    (* first : copy in the 3.2 contents *)
    Hashtbl.iter add_elem dtd32.contents;

    (* frameset and frames is pretty simple *)
    add_elem "frameset" (sol ["frameset"; "frame"; "noframes"]);
    add_elem "frame" Elements.empty;
    omit_close "frame";
    (* we say that noframes contains the same thing as body in 3.2 *)
    add_elem "noframes" (Hashtbl.find dtd.contents "body");
    (* and we say that frameset can occur in html *)
    let html_contents = Hashtbl.find dtd.contents "html" in
    Hashtbl.remove dtd.contents "html";
    add_elem "html" (Elements.add "frameset"
               (Elements.add "noframes" html_contents));
    dtd
```

⟨*toplevel* Dtd._2 239a⟩≡ (293a)
```
  let _ = add dtd32f
```

20.9 Fake CGI

⟨File.request() *if CGI path* 239b⟩≡ (210f)
```
  if is_cgi path
  then (fake_cgi wr cont path; (fun () -> ()))
```

⟨*constant* File.binary_path 239c⟩≡ (316g)
```
  (* Pref stuff *)
  let binary_path = ref ([] : string list)
```

⟨*function* File.is_cgi 239d⟩≡ (316g)
```
  let is_cgi file =
    match !binary_path with
    | [] -> false
    | path ->
        let l = String.length file in
        List.exists (fun dir ->
      let ldir = String.length dir in
      l > ldir && String.sub file 0 ldir = dir)
      path
  (*
   * Display a file on the local unix file system (file:)
   *  is path really supposed to be absolute ?
   * Note: completely ignores method (GET, POST,...)
   *)
```

⟨*function* File.fake_cgi 239e⟩≡ (316g)
```
  (* Not true CGI interface, just a hack *)
  (* TODO: headers ? *)
  let fake_cgi wwwr cont path =
    try
      let (cmd_in, cmd_out) = pipe() in
      let cmd, args =
        try
         let pos = String.index path '?' in
         let cmd = String.sub path 0 pos in
         if pos + 1 = String.length path
         then cmd, [| cmd |]
         else cmd, [|cmd; String.sub path (pos+1) (String.length path - pos - 1)|]
        with Not_found -> path, [| path |]
      in
      match Low.fork() with
      (* child *)
      | 0 ->
          close cmd_in;
          dup2 cmd_out stdout; close cmd_out;
          begin
            try execvp cmd args
            with Unix_error(e, _, _) ->
              Munix.write_string stdout "HTTP/1.0 404 Not found\r\n";
              Munix.write_string stdout "Content-Type: text/html\r\n\r\n";
              Munix.write_string stdout "<H1>Cannot execute local file</H1>";
              Munix.write_string stdout "Command \"";
              Munix.write_string stdout cmd;
              Munix.write_string stdout "\" failed:";
            Munix.write_string stdout (Unix.error_message e);
            Munix.write_string stdout "\n";
            (* nosemgrep: do-not-use-exit *)
            exit 1
         end
     (* parent *)
     | _n ->
       close cmd_out;
       let dh = {document_id = document_id wwwr;
                 document_referer = wwwr.www_link.h_context;
                 document_status = 0;
                 dh_headers = [];
                 document_feed = Feed.make_feed cmd_in (Low.count_read (Unix.read cmd_in));
                 document_fragment = wwwr.www_fragment;
                 document_logger = Document.tty_logger}
       in
       dh.document_feed.feed_schedule
         (fun () ->
            try
             if dh.dh_headers = [] then begin
             (* it should be the HTTP Status-Line *)
             let l = Low.read_line cmd_in in
               dh.document_status <- (Http_headers.parse_status l).status_code;
               dh.dh_headers <- [l] (* keep it there *)
             end
           else
             dh.dh_headers <-
               read_headers (Low.read cmd_in) dh.dh_headers
           with
            | End_of_headers ->
                dh.document_feed.feed_unschedule();
                cont.document_process dh
            | Not_found -> (* No HTTP/ header *)
                dh.document_feed.feed_unschedule();
                dh.document_status <- 200;
                dh.dh_headers <- ["Content-Type: text/plain"];
                cont.document_process dh
            | Unix_error(_,_,_) ->
                dclose true dh;
                raise (File_error (s_
                   "Error while reading headers of %s\n%s" path "(read)"))
            | Http_headers.Invalid_header s ->
                dclose true dh;
                raise (File_error (s_
                       "Error while reading headers of %s\n%s" path s))
            | End_of_file ->
                dclose true dh;
                raise (File_error (s_
                   "Error while reading headers of %s\n%s" path "eof"))
           )
    with Unix_error(_,_,_) ->
      raise (File_error (s_ "cannot exec file"))
```

20.10 Signals

⟨Main.main() *signal handling* 240⟩≡ (29c)

```
  Sys.catch_break true;
  (* Avoid SIGPIPE completely, in favor of write() errors *)
```

```
Sys.set_signal Sys.sigpipe Sys.Signal_ignore;
```

Chapter 21

Conclusion

Appendix A

Debugging

⟨*signature* Log.debug_mode 243a⟩≡ (274c)
```
  val debug_mode : bool ref
```

⟨*constant* Log.debug_mode 243b⟩≡ (275a)
```
  let debug_mode = ref true
```

A.1 Logging

⟨*signature* Log.f 243c⟩≡ (274c)
```
  val f : string -> unit
```

⟨*function* Log.f 243d⟩≡ (275a)
```
  (* flushes ! *)
  let f s =
    try prerr_endline s
    with _ -> ()
```

⟨*signature* Log.debug 243e⟩≡ (274c)
```
  val debug : string -> unit
```

⟨*function* Log.debug 243f⟩≡ (275a)
```
  let debug s =
    if !debug_mode
    then f s
```

A.2 Postmortem

⟨Main.main() *after event loop, if debug mode* 243g⟩≡ (244a 29c)
```
  if !Log.debug_mode then begin
    Cache.postmortem();
    Gcache.postmortem()
  end;
```

⟨*toplevel* Main._1 243h⟩≡ (463b)
```
  let _ =
    Cap.main (fun (caps : Cap.all_caps) ->
      let argv = CapSys.argv caps in
      Exit.exit caps (Exit.catch (fun () -> postmortem caps argv)))
```

⟨*function* Main.postmortem 244a⟩≡ (463b)
```
  let postmortem (caps: < caps; Cap.stdout; Cap.stderr; ..>)
                 (argv : string array) : Exit.t =
    try
      main caps argv
    with
    | Dynlink.Error err ->
        Logs.err (fun m -> m "dynlink error = %s" (Dynlink.error_message err));
        Exit.Code 1
    | Failure s ->
        Logs.err (fun m -> m "mmm: %s" s);
        Exit.Code 2
    | e ->
        ⟨Main.main() after event loop, if debug mode 243g⟩
        raise e
```

⟨*signature* Cache.postmortem 244b⟩≡ (309l)
```
  val postmortem : unit -> unit
```

⟨*function* Cache.postmortem 244c⟩≡ (314e)
```
  (* Debugging *)
  let postmortem () =
    Log.f (sprintf "Cache size(max): %d(%d)" !current !max_documents);
    !memory |> List.iter (fun (did, entry) ->
      Log.f (sprintf "%s(%d) %s"
              (Url.string_of did.document_url)
              did.document_stamp
              (match entry.cache_document.document_data with
               | MemoryData _ -> "in memory"
               | FileData (f,true) -> !!f
               | FileData (f,false) -> "fake " ^ !!f)
               );
      entry.cache_document.document_headers
      |> List.rev |> List.iter (fun h -> Log.f (sprintf "%s" h));

      if entry.cache_pending
      then Log.f "pending ";

      Log.f (sprintf "Last used: %f" entry.cache_lastused);
      Log.f ""
    )
```

A.3 Subsystems

A.3.1 Requests

⟨Www.request *logging method* 244d⟩≡ (18f)
```
  mutable www_logging : string -> unit;   (* logging *)
```

A.3.2 Documents

⟨Document.handle *other fields* 244e⟩+≡ (22b) ◁229b
```
  mutable document_logger : logger;
    (* how to log information relative to this document processing *)
```

⟨*signature type Document.logger* 245a⟩≡ (291b)
```
  (* pad: exported for tk_document, but normally should be abstract *)
  type logger = {
    logger_destroy : bool -> unit;
    logger_progress : int -> unit;
    logger_msg : string -> unit;
    logger_end : string -> unit
  }
```

⟨*signature* Document.tty_logger 245b⟩≡ (291b)
```
  val tty_logger : logger
```

⟨*constant* Document.tty_logger 245c⟩≡ (291c)
```
  let tty_logger =
    { logger_destroy = (fun _ -> ());
      logger_progress = (fun _ -> ());
      logger_msg = Log.f;
      logger_end = Log.f
     }
```

A.3.3 HTML

⟨*signature* Html.verbose 245d⟩≡ (293d)
```
  val verbose : bool ref
    (* verbose mode for HTML related stuff *)
```

⟨*constant* Html.verbose 245e⟩≡ (295)
```
  let verbose = ref false
```

⟨Htparse.html_lex() *print token t if verbose* 245f⟩≡ (82d)
```
  if !verbose
  then begin
     Html.print t;
     flush stdout
  end
```

⟨*signature* Html.warning 245g⟩≡ (293d)
```
  val warning : string -> location -> unit
```

⟨*function* Html.warning 245h⟩≡ (295)
```
  let warning s (Loc(n,m)) =
    if !verbose then begin
      eprintf "HTML Warning: %s at (%d, %d)\n" s n m;
      flush stderr
     end
```

⟨*signature* Html_eval.debug 245i⟩≡ (297a)
```
  (* HTML Evaluation *)
  val debug : bool ref
```

⟨*constant* Html_eval.debug 245j⟩≡ (297b)
```
  let debug = ref false
```

A.3.4 HTTP

⟨*constant* Http.verbose 245k⟩≡ (306d)
```
  let verbose = ref false
```

⟨Http.async_request() *log request string req if verbose* 246a⟩≡ (102d)
```
  if !verbose
  then Logs.debug (fun m -> m "%s" req);
```

A.3.5 Caches

⟨*signature* Cache.debug 246b⟩≡ (309l)
```
  (* Configurable settings *)
  val debug : bool ref
```

⟨*constant* Cache.debug 246c⟩≡ (314e)
```
  let debug = ref false
```

A.3.6 Viewer

⟨Viewers.context *logging methods signatures* 246d⟩≡ (24a)
```
  method virtual log : string -> unit
```

A.3.7 Scheduler

⟨*signature* Scheduler.debug 246e⟩≡ (323)
```
  val debug : bool ref
```

⟨*constant* Scheduler.debug 246f⟩≡ (324)
```
  let debug = ref false
```

A.3.8 HTML display

⟨*signature* Html_disp.verbose 246g⟩≡ (423a)
```
  val verbose : bool ref
```

⟨*constant* Html_disp.verbose 246h⟩≡ (423b)
```
  let verbose = ref false
```

A.3.9 GUI

⟨*function* Debug.init 246i⟩≡ (448a)
```
  let init () =
    Frx_rpc.register "cb" active_cb;
    Frx_rpc.register "cache"
       (fun _ ->
         Cache.postmortem();
         Gcache.postmortem();
         flush stderr);
    Frx_rpc.register "images" (fun _ ->
      Img.ImageData.dump();
      flush stderr);
    Frx_rpc.register "camltkdb" (fun _ ->
      Protocol.debug := not !Protocol.debug)
```

⟨*function* Debug.active_cb 247a⟩≡ (448a)
```
let active_cb _ =
  let cnter = ref 0 in
  Hashtbl.iter
    (fun w id ->
      incr cnter;
      Printf.fprintf stdout "%s %s %b\n"
        (Widget.name w) (string_of_cbid id) (Winfo.exists w)
    )
    callback_memo_table;
  Printf.fprintf stdout "Memo cb: %d\n" !cnter;
  cnter := 0;
  Hashtbl.iter (fun _ _ -> incr cnter) callback_naming_table;
  Printf.fprintf stdout "Active cb: %d\n" !cnter;
  flush stdout
```

A.4 Dumpers

A.4.1 URLs

⟨*signature* Url.string_of_protocol 247b⟩≡ (286d)
```
val string_of_protocol: protocol -> string
  (* maps FTP to "ftp", etc... *)
```

⟨*function* Url.string_of_protocol 247c⟩≡ (287a)
```
let string_of_protocol = function
 | FTP -> "ftp"
 | HTTP -> "http"
 | HTTPS -> "https"
 | GOPHER -> "gopher"
 | MAILTO -> "mailto"
 | NEWS -> "news"
 | NNTP -> "nntp"
 | TELNET -> "telnet"
 | WAIS -> "wait"
 | FILE -> "file"
 | PROSPERO -> "prospero"
 | OtherProtocol s -> s
```

⟨*signature* Url.string_of 247d⟩≡ (286d)
```
(* These are used to get "normalized urls" *)
val string_of: t -> string
```

⟨*function* Url.string_of 247e⟩≡ (287a)
```
let string_of (p : t) : string =
  let buf = Ebuffer.create 128 in
  let ws x = Ebuffer.output_string buf x in
  let wc x = Ebuffer.output_char buf x in
  let write_userpass () =
      match p.user, p.password with
         None, None -> ()
       | Some u, Some p -> ws u; wc ':'; ws p; wc '@'
       | Some u, None ->   ws u; wc ':'; wc '@'
       | None, Some _ -> failwith "url_of_parsed"
  in
  (* hostname is always put in lowercase *)
  let write_hostport def =
      match p.host, p.port with
```

```
       None, None -> ()
     | Some h, None -> ws (String.lowercase_ascii h)
     | Some h, Some p when p = def -> ws (String.lowercase_ascii h)
     | Some h, Some p ->
         ws (String.lowercase_ascii h); wc ':'; ws (string_of_int p)
     | None, Some _ -> failwith "url_of_parsed"
in
let write_pathsearch () =
    match p.path, p.search with
      None, None -> wc '/'
    | Some p, Some s -> wc '/'; ws p; wc '?'; ws s
    | Some p, None -> wc '/'; ws p
    | None, Some _ -> failwith "url_of_parsed"
in
let write_slashpath () =
    match p.path with
     None -> ()
    | Some p -> wc '/'; ws p
in
let write_path () =
    match p.path with
     None -> ()
    | Some p -> ws p
in
let write_fhost () =
    match p.host with
     None -> ws "localhost"
    | Some h -> ws (String.lowercase_ascii h)
in
begin match p.protocol with
  FTP ->
    ws "ftp://"; write_userpass (); write_hostport 21; write_slashpath ()
| HTTP ->
    ws "http://"; write_hostport 80; write_pathsearch ()
| HTTPS ->
    ws "https://"; write_hostport 443; write_pathsearch ()
| GOPHER ->
    ws "gopher://"; write_hostport 70; write_slashpath ()
| MAILTO -> ws "mailto:"; write_path()
| NEWS -> ws "news:"; write_path()
| NNTP -> ws "nntp:"; write_hostport 119; write_path()
| TELNET -> ws "telnet://"; write_userpass(); write_hostport 23
| WAIS -> ws "wais://"; write_hostport 210; write_pathsearch()
| FILE ->
  (* for file: we have to transform to ftp: if host is not localhost *)
  begin match p.host with
    None | Some "localhost" ->
      ws "file://"; write_fhost(); write_slashpath()
  | Some _h ->
     p.protocol <- FTP;
      ws "ftp://"; write_userpass (); write_hostport 21; write_slashpath ()
  end
| PROSPERO -> ws "prospero://"; write_hostport 1525; write_slashpath()
| OtherProtocol s -> ws s; ws ":"; write_path()
end;
Ebuffer.get buf
```

A.4.2 Links

⟨*signature* Hyper.string_of 249a⟩≡ (288c)
```
  val string_of : link -> string
    (* make an absolute URI (including fragment) from link
       raises Invalid_link(msg) *)
```

⟨*function* Hyper.string_of 249b⟩≡ (289a)
```
  let string_of link =
    let uri = resolve link in
     match uri.uri_fragment with
        None -> uri.uri_url
      | Some f -> Printf.sprintf "%s#%s" uri.uri_url f
```

A.4.3 DTDs

⟨*signature* Dtd.dump 249c⟩≡ (292)
```
  val dump : t -> unit
```

⟨*function* Dtd.dump 249d⟩≡ (293a)
```
  let dump dtd =
    dtd.contents |> Hashtbl.iter (fun s contents ->
        printf "Element %s %s %s\n" s
               (if Elements.mem s dtd.open_omitted then "O" else "-")
               (if Elements.mem s dtd.close_omitted then "O" else "-");
        printf "Contains:";
        contents |> Elements.iter (fun e -> printf " %s" e);
        printf "\n"
    )
```

A.4.4 HTML

⟨*signature* Html.print 249e⟩≡ (293d)
```
  val print : token -> unit
    (* for debugging, prints an HTML token *)
```

⟨*function* Html.print 249f⟩≡ (295)
```
  let print = function
      PCData s -> eprintf "PCData: %s\n" s
    | CData s -> eprintf "CData: %s\n" s
    | OpenTag {tag_name = n; attributes = l} ->
              eprintf "Open: %s\n" n;
           List.iter (function (a,v) ->
                     eprintf "%s=%s\n" a v) l
    | CloseTag n -> eprintf "Close: %s\n" n
    | Comment s -> eprintf "Comment: %s\n" s
    | Doctype s -> eprintf "Doctype: %s\n" s
    | EOF -> eprintf "EOF\n"
```

Appendix B

Profiling

Appendix C

Error Management

C.1 Error.t

⟨*signature* Error.f 251a⟩≡ (266a)

```
  val f : string -> unit
```

⟨*signature* Error.ok 251b⟩≡ (266a)

```
  val ok : string -> unit
```

⟨*signature* Error.choose 251c⟩≡ (266a)

```
  val choose : string -> bool
```

⟨*signature* Error.ari 251d⟩≡ (266a)

```
  val ari : string -> int
```

⟨*signature class* Error.t 251e⟩≡ (266a)

```
  class virtual t : object
   method virtual f : string -> unit
   method virtual ok : string -> unit
   method virtual choose : string -> bool
   method virtual ari : string -> int
  end
```

⟨*signature* Error.default 251f⟩≡ (266a)

```
  val default : t ref
```

⟨*constant* Error.default 251g⟩≡ (266b)

```
  let default = ref (new x)
```

⟨*functions* Error.xxx 251h⟩≡ (266b)

```
  let f msg =
    !default#f msg
  let ok msg =
    !default#ok msg
  let choose msg =
    !default#choose msg
  let ari msg =
    !default#ari msg
```

⟨*class* Error.t 251i⟩≡ (266b)

C.2 Subsystems

C.2.1 Requests

⟨Www.request *error managment method* 252a⟩≡ (18f)
```
  mutable www_error : Error.t;
```

C.3 Exceptions

C.3.1 URL

⟨*exception* Url.Url_Lexing 252b⟩≡ (287a 286d)
```
  exception Url_Lexing of string * int
```

⟨*exception* Url.Invalid_url 252c⟩≡ (287a)
```
  (*exception Invalid_url of t * string*)
```

C.3.2 Links

⟨*exception* Hyper.Invalid_link 252d⟩≡ (289a 288c)
```
  exception Invalid_link of link_error
```

⟨*type* Hyper.link_error 252e⟩≡ (289a 288c)
```
  type link_error =
    | LinkResolve of string
    | UrlLexing of string * int
```

C.3.3 Web requests

⟨*exception* Www.Invalid_request 252f⟩≡ (291a 290b)
```
  exception Invalid_request of request * string
```

C.3.4 HTML

⟨*exception* Html.Html_Lexing 252g⟩≡ (295 293d)
```
  exception Html_Lexing of string * int
```

⟨*exception* Html.Invalid_Html 252h⟩≡ (295 293d)
```
  exception Invalid_Html of string
```

C.3.5 HTTP

⟨*exception* Http_headers.Invalid_HTTP_header 252i⟩≡ (302 300g)
```
  exception Invalid_header of string
```

⟨*exception* Http.HTTP_error 252j⟩≡ (306)
```
  exception HTTP_error of string
```

Appendix D

A Preferences Library

Appendix E

Standard Library

E.1 Lists

⟨*signature* Mlist.hdn 254a⟩≡ (279a)
```
  (* List utilities *)
  val hdn : 'a list -> int -> 'a list
     (* [hdn [a1;a2;...;an;...; ak] returns [a1;a2;...;an] *)
```

⟨*signature* Mlist.tln 254b⟩≡ (279a)
```
  val tln : 'a list -> int -> 'a list
     (* [tln [a1;a2;...;an;...; ak] returns [an+1;...; ak] *)
```

⟨*signature* Mlist.except_assoc 254c⟩≡ (279a)
```
  val except_assoc: 'a -> ('a * 'b) list -> ('a * 'b) list
```

⟨*signature* Mlist.exceptq 254d⟩≡ (279a)
```
  val exceptq: 'a -> 'a list -> 'a list
```

⟨*signature* Mlist.rev_do_list 254e⟩≡ (279a)
```
  val rev_do_list : ('a -> unit) -> 'a list -> unit
```

⟨*signature* Mlist.do_listi 254f⟩≡ (279a)
```
  val do_listi : (int -> 'a -> unit) -> int -> 'a list -> unit
```

⟨*function* Mlist.tln 254g⟩≡ (279b)
```
  (* tln l n *)
  let rec tln l = function
     0 -> l
   | n -> if l = [] then [] else tln (List.tl l) (pred n)
```

⟨*function* Mlist.hdn 254h⟩≡ (279b)
```
  let hdn l =
    let rec h l acc = function
      0 -> List.rev acc
    | n -> if l = [] then List.rev acc
          else h (List.tl l) (List.hd l :: acc) (pred n) in
    h l []
```

⟨*function* Mlist.except_assoc 254i⟩≡ (279b)
```
  let except_assoc x =
    let rec ex acc = function
        [] -> acc
      | (y,_v)::l when x = y -> ex acc l
      | z :: l -> ex (z::acc) l
    in
    ex []
```

⟨*function* Mlist.exceptq 255a⟩≡ (279b)
```
let exceptq x =
  let rec ex acc = function
      [] -> acc
   | y::l when y == x -> ex acc l
   | y::l -> ex (y::acc) l
  in
  ex []
```

⟨*function* Mlist.rev_do_list 255b⟩≡ (279b)
```
(* List.iter from right to left *)
let rev_do_list f =
 let rec do_list_f = function
     [] -> () | x::l -> do_list_f l; f x in
  do_list_f
```

⟨*function* Mlist.do_listi 255c⟩≡ (279b)
```
let rec do_listi f n l =
  match l with
    [] -> ()
  | (x::l) -> f n x; do_listi f (succ n) l
```

E.2 Strings

⟨*signature* Mstring.split_str 255d⟩≡ (279c)
```
(* String utilities *)
val split_str : (char -> bool) -> string -> string list
```

⟨*signature* Mstring.get_suffix 255e⟩≡ (279c)
```
val get_suffix : string -> string
```

⟨*signature* Mstring.hex_to_dec 255f⟩≡ (279c)
```
val hex_to_dec : char -> int
```

⟨*signature* Mstring.dec_to_hex 255g⟩≡ (279c)
```
val dec_to_hex : int -> char
```

⟨*signature* Mstring.hex_to_string 255h⟩≡ (279c)
```
val hex_to_string : string -> string
```

⟨*signature* Mstring.gensym 255i⟩≡ (279c)
```
val gensym : string -> string
```

⟨*signature* Mstring.egensym 255j⟩≡ (279c)
```
val egensym : string -> unit -> string
```

⟨*signature* Mstring.rem_trailing_sp 255k⟩≡ (279c)
```
val rem_trailing_sp : string -> string
```

⟨*signature* Mstring.catenate_sep 255l⟩≡ (279c)
```
val catenate_sep : string -> string list -> string
```

⟨*signature* Mstring.norm_crlf 255m⟩≡ (279c)
```
val norm_crlf : bool -> string -> int -> int -> string * bool
    (* [norm_crlf last_was_cr buf offs len] returns
       buf with CRLF/CR/LF converted to LF, and a flag indicating
       whether last char was CR *)
```

⟨*function* Mstring.split_str 256a⟩≡ (280a)
```
  (* split a string according to char_sep predicate *)
  let split_str char_sep str =
    let len = String.length str in
    if len = 0 then [] else
      let rec skip_sep cur =
        if cur >= len then cur
        else if char_sep str.[cur] then skip_sep (succ cur)
        else cur  in
      let rec split beg cur =
        if cur >= len then
      if beg = cur then []
      else [String.sub str beg (len - beg)]
        else if char_sep str.[cur]
         then
           let nextw = skip_sep cur in
            (String.sub str beg (cur - beg))
          ::(split nextw nextw)
         else split beg (succ cur) in
      let wstart = skip_sep 0 in
      split wstart wstart
```

⟨*function* Mstring.get_suffix 256b⟩≡ (280a)
```
  (* extract the . suffix (dot excluded) of a string *)
  let get_suffix s =
    try
      let dotpos = succ (String.rindex s '.') in
        String.sub s dotpos (String.length s - dotpos)
    with
      Not_found -> ""
```

⟨*function* Mstring.hex_to_dec 256c⟩≡ (280a)
```
  (* HEX/DEC conversions *)
  let hex_to_dec c = match c with
      '0'..'9' -> Char.code c - 48
    | 'a'..'f' -> Char.code c - 87 (* 87 = Char.code 'a' - 10 *)
    | 'A'..'F' -> Char.code c - 55 (* 55 = Char.code 'A' - 10 *)
    | _ -> failwith "hex_to_dec"
```

⟨*function* Mstring.dec_to_hex 256d⟩≡ (280a)
```
  let dec_to_hex i =
    if i < 10 then Char.chr (i + 48)  (* 48 = Char.code '0' *)
    else Char.chr (i + 55)            (* 55 = Char.code 'A' - 10 *)
```

⟨*function* Mstring.hex_to_string 256e⟩≡ (280a)
```
  (* Converting a hex stored string *)
  let hex_to_string s =
    let len = String.length s / 2 in
    let res = Bytes.create len in
      for i = 0 to len - 1 do
        Bytes.set res i (Char.chr ( 16 * (hex_to_dec s.[i+i]) + hex_to_dec s.[i+i+1]));
        done;
      Bytes.to_string res
```

⟨*constant* Mstring.gensym 256f⟩≡ (280a)
```
  let gensym =
    let cnter = ref 0 in
    (fun n ->
      incr cnter;
      n ^ string_of_int !cnter)
```

⟨*function* Mstring.egensym 257a⟩≡ (280a)
```
let egensym s =
  let cnter = ref 0 in
  (fun () ->
    incr cnter;
    s ^ string_of_int !cnter)
```

⟨*function* Mstring.rem_trailing_sp 257b⟩≡ (280a)
```
let rem_trailing_sp s =
  let l = String.length s in
  let pos = ref (l - 1) in
  while !pos >= 0 && List.mem s.[!pos] [' '; '\t'] do decr pos done;
  if !pos = l - 1 then s
  else String.sub s 0 (succ !pos)
```

⟨*function* Mstring.catenate_sep 257c⟩≡ (280a)
```
let catenate_sep sep =
  function
      [] -> ""
    | x::l -> List.fold_left (fun s s' -> s^sep^s') x l
```

⟨*function* Mstring.norm_crlf 257d⟩≡ (280a)
```
(* Filters CRLF:
 *  CR -> LF
 *  CRLF -> LF
 *  LF -> LF
 * We do this on successive chunks of a stream, so we need to consider
 * the case when the chunk finishes on CR.
 * Assume len > 0
 *)

let norm_crlf lastwascr buf offs len =
  let rpos = ref offs
  and wpos = ref 0
  and dest = Bytes.create (len + 1) (* we need one more char *)
  and limit = offs + len - 1
  and lastiscr = ref false in
  if lastwascr then
    if buf.[!rpos] = '\n' then begin
      Bytes.set dest !wpos '\n';
      incr rpos; incr wpos
    end
    else begin
      Bytes.set dest !wpos '\n'; incr wpos
    end;

  while !rpos < limit do
    match buf.[!rpos] with
      '\n' -> Bytes.set dest !wpos '\n'; incr rpos; incr wpos
    | '\r' ->
    if buf.[!rpos + 1] = '\n'
    then begin Bytes.set dest !wpos '\n'; rpos := !rpos + 2; incr wpos end
    else begin Bytes.set dest !wpos '\n'; incr rpos; incr wpos end
    | c -> Bytes.set dest !wpos c; incr rpos; incr wpos
  done;
  begin match buf.[offs+len-1] with
    '\n' -> Bytes.set dest !wpos '\n'; incr wpos
  | '\r' -> lastiscr := true
  | c -> Bytes.set dest !wpos c; incr wpos
  end;
  Bytes.sub_string dest 0 !wpos, !lastiscr
```

E.3 Extensible buffers

⟨*signature* Ebuffer.create 258a⟩≡ (265b)
```
  val create : int -> t
    (* [create n] creates a buffer with initial size [n] *)
```

⟨*signature* Ebuffer.output_string 258b⟩≡ (265b)
```
  val output_string : t -> string -> unit
    (* [output_string buf s] appends [s] to [buf] *)
```

⟨*signature* Ebuffer.output_char 258c⟩≡ (265b)
```
  val output_char : t -> char -> unit
    (* [output_char buf c] appends [c] to [buf] *)
```

⟨*signature* Ebuffer.output 258d⟩≡ (265b)
```
  val output : t -> string -> int -> int -> unit
    (* [output buf s offs len] appends [len] characters of [s], starting
       at offset [offs] to [buf].
       Raises [Invalid_argument] if [offs] and [len] do not designate a
       valid substring of [s] *)
```

⟨*signature* Ebuffer.get 258e⟩≡ (265b)
```
  val get : t -> string
    (* [get buf] returns the current contents of [buf] *)
```

⟨*signature* Ebuffer.used 258f⟩≡ (265b)
```
  val used : t -> int
    (* [used buf] returns the current length of [buf] *)
```

⟨*signature* Ebuffer.reset 258g⟩≡ (265b)
```
  val reset : t -> unit
    (* [reset buf] emties [buf] *)
```

⟨*signature* Ebuffer.t *abstract* 258h⟩≡ (265b)
```
  type t
```

⟨*type* Ebuffer.t 258i⟩≡ (265c)
```
  (* Extensible buffers *)
  type t = {
      mutable buffer : bytes;
      mutable pos : int;
      mutable len : int}
```

⟨*function* Ebuffer.create 258j⟩≡ (265c)
```
  let create n = {
     buffer = Bytes.create n;
     pos = 0;
     len = n
     }
```

⟨*function* Ebuffer.reset 258k⟩≡ (265c)
```
  let reset buf =
      buf.pos <- 0
```

⟨*function* Ebuffer.newsize 258l⟩≡ (265c)
```
  let newsize old added =
    if added < old then old + old
    else old + old + added
```

⟨*function* Ebuffer.output_string 259a⟩≡ (265c)
```
  let output_string buf s =
    let l = String.length s in
    if buf.pos + l > buf.len then begin
      let size = newsize buf.len l in
      let news = Bytes.create size in
        Bytes.unsafe_blit buf.buffer 0 news 0 buf.pos;
        buf.buffer <- news;
        buf.len <- size
      end;
    Bytes.unsafe_blit_string s 0 buf.buffer buf.pos l;
    buf.pos <- buf.pos + l
```

⟨*function* Ebuffer.output_char 259b⟩≡ (265c)
```
  let output_char buf c =
    if buf.pos >= buf.len then begin
      let size = newsize buf.len 1 in
      let news = Bytes.create size in
        Bytes.unsafe_blit buf.buffer 0 news 0 buf.pos;
        buf.buffer <- news;
        buf.len <- size
      end;
    Bytes.set buf.buffer buf.pos c;
    buf.pos <- buf.pos + 1
```

⟨*function* Ebuffer.output 259c⟩≡ (265c)
```
  let output buf s ofs l =
    if buf.pos + l > buf.len then begin
      let size = newsize buf.len l in
      let news = Bytes.create size in
        Bytes.unsafe_blit buf.buffer 0 news 0 buf.pos;
        buf.buffer <- news;
        buf.len <- size
      end;
    String.blit s ofs buf.buffer buf.pos l;
    buf.pos <- buf.pos + l
```

⟨*function* Ebuffer.get 259d⟩≡ (265c)
```
  let get buf =
    Bytes.sub_string buf.buffer 0 buf.pos
```

⟨*function* Ebuffer.used 259e⟩≡ (265c)
```
  let used buf =
    buf.pos
```

E.4 Files

⟨*signature* Msys.tilde_subst 259f⟩≡ (280b)
```
  val tilde_subst : string -> string
      (* substitute ~ at beginning of file path *)
```

⟨*signature* Msys.rm 259g⟩≡ (280b)
```
  val rm: string -> unit
      (* quiet unlink *)
```

⟨*signature* Msys.fsize 259h⟩≡ (280b)
```
  val fsize: string -> int
      (* file size *)
```

⟨*signature* Msys.mktemp 260a⟩≡ (280b)
```
  val mktemp : string -> string
```

⟨*function* Msys.next_slash 260b⟩≡ (281a)
```
  (* skip to next / *)
  let rec next_slash s n =
    if  n >= String.length s || s.[n] = '/'
    then n
    else next_slash s (succ n)
```

⟨*function* Msys.tilde_subst 260c⟩≡ (281a)
```
  let tilde_subst s =
   try
    if s = "" || s.[0] <> '~' then s
    else
      let len = String.length s in
      if len = 1 then Sys.getenv "HOME"
      else match s.[1] with
        '/' ->
         Filename.concat (Sys.getenv "HOME") (String.sub s 2 (len - 2))
       | _ ->
        let final = next_slash s 1 in
        let user = String.sub s 1 (pred final) in
        let pwnam = getpwnam user in
          if succ final >= len then pwnam.pw_dir
          else
           Filename.concat pwnam.pw_dir
                (String.sub s (succ final) (len - (succ final)))
   with
      Unix_error(_,_,_) -> s
    | Sys_error _ -> s
    | Not_found -> s
```

⟨*function* Msys.rm 260d⟩≡ (281a)
```
  (* Quiet unlink *)
  let rm s = try unlink s with Unix_error _ -> ()
```

⟨*function* Msys.rmdir 260e⟩≡ (281a)
```
  let rmdir dir =
    try
      let dh = opendir dir
      and l = ref [] in
      try while true do
        let f = readdir dh in
        if f <> "." && f <> ".." then l := f :: !l
      done
      with
        End_of_file ->
      closedir dh;
      List.iter (fun f -> rm (Filename.concat dir f)) !l;
      Unix.rmdir dir
    with
      Unix_error _ -> ()
```

⟨*function* Msys.fsize 260f⟩≡ (281a)
```
  let fsize f =
    try (Unix.stat f).st_size
    with Unix_error(_,_,_) -> raise Not_found
```

⟨*constant* Msys.tmp_dir 260g⟩≡ (281a)
```
  let tmp_dir = ref "/tmp"
```

⟨*constant* Msys.mktemp 261a⟩≡ (281a)
```
  (* We know use our own private directory in /tmp, cleared at exit-time,
     so no one can snoop our temporary files *)
  let mktemp =
    let cnter = ref 0
    and pid = Unix.getpid()
    and id = ref 0 in
    let thisdir =
      let testdir = ref "" in
      try while true do
        testdir := Filename.concat !tmp_dir ("mmm" ^ string_of_int pid
                             ^ "_" ^ string_of_int !id);
        if not (Sys.file_exists !testdir) then raise Exit;
        incr id;
        if !id >= 16 then
      raise (Failure ("Too many MMM temporary directory in " ^ !tmp_dir ^
              ". Clean them first."))
      done; "" (* cannot reach *)
      with
        Exit -> !testdir
    in
    Unix.mkdir thisdir 0o700;
    at_exit (fun () -> rmdir thisdir);
    (function prefx ->
        incr cnter;
        (Filename.concat thisdir (prefx ^ string_of_int !cnter)))
```

E.5 Dates

⟨*signature* Date.asc_wkday 261b⟩≡ (264c)
```
  val asc_wkday : int -> string
      (* [asc_wkday n] maps 0..6 to Sun..Sat *)
```

⟨*signature* Date.asc_month 261c⟩≡ (264c)
```
  val asc_month : int -> string
      (* [asc_month n] maps 0..11 to Jan..Dec *)
```

⟨*signature* Date.asc 261d⟩≡ (264c)
```
  val asc : float -> string
      (* [asc uxtime] RFC822 of unix time *)
```

⟨*signature* Date.asc_now 261e⟩≡ (264c)
```
  val asc_now : unit -> string
      (* [asc_now ()] RFC822 of now *)
```

⟨*signature* Date.commonlog 261f⟩≡ (264c)
```
  val commonlog : float -> string
    (* Text version (Common log format) of an Unix time value *)
```

⟨*signature* Date.compare_time 261g⟩≡ (264c)
```
  val compare_time : int list * int list -> int
      (* [compare_time l1 l2] compare lists encodings of timestamps
         Encoding must be:
          [year; month; mday; hour; min; sec]
       *)
```

⟨*function* Date.asc_wkday 262a⟩≡ (265a)

```
  let asc_wkday = function
     0 -> "Sun"
   | 1 -> "Mon"
   | 2 -> "Tue"
   | 3 -> "Wed"
   | 4 -> "Thu"
   | 5 -> "Fri"
   | 6 -> "Sat"
   | _ -> assert false
```

⟨*function* Date.asc_month 262b⟩≡ (265a)

```
  let asc_month = function
     0 -> "Jan"
   | 1 -> "Feb"
   | 2 -> "Mar"
   | 3 -> "Apr"
   | 4 -> "May"
   | 5 -> "Jun"
   | 6 -> "Jul"
   | 7 -> "Aug"
   | 8 -> "Sep"
   | 9 -> "Oct"
   | 10 -> "Nov"
   | 11 -> "Dec"
   | _ -> assert false
```

⟨*function* Date.asc 262c⟩≡ (265a)

```
  (* Produces RFC822 style *)
  let asc ut =
    let tm = gmtime ut in
      sprintf "%s, %02d %s %d %02d:%02d:%02d GMT"
          (asc_wkday tm.tm_wday)
      tm.tm_mday
      (asc_month tm.tm_mon)
      (tm.tm_year + 1900)
      tm.tm_hour
      tm.tm_min
      tm.tm_sec
```

⟨*function* Date.asc_now 262d⟩≡ (265a)

```
  let asc_now () = asc (time())
```

⟨*function* Date.commonlog 262e⟩≡ (265a)

```
  (* Timezone ??? *)
  let commonlog int =
    let tm = localtime int in
    sprintf "%02d/%s/%d:%02d:%02d:%02d"
        tm.tm_mday
        (asc_month tm.tm_mon)
        (tm.tm_year + 1900)
        tm.tm_hour
        tm.tm_min
        tm.tm_sec
```

⟨*function* Date.compare_time 262f⟩≡ (265a)

```
  let rec compare_time = function
     [], [] -> 0
   | (x::xx), (y::yy) when x = y -> compare_time (xx, yy)
   | (x::_), (y::_) when x < y -> -1
   | (x::_), (y::_) when x > y -> 1
   |  _, _ -> assert false
```

Appendix F

Extra Code

⟨www/lexurl.ml 263a⟩≡

⟨http/lexheaders.ml 263b⟩≡

⟨http/lexdate.ml 263c⟩≡

⟨html/lexhtml.ml 263d⟩≡

⟨commons/timer_.ml 263e⟩≡

⟨commons/fileevent_.ml 263f⟩≡

F.1 core/

F.1.1 core/condition.mli

⟨*signature* Condition.create 263g⟩≡ (263k)
```
  val create : unit -> t
```

⟨*signature* Condition.wait 263h⟩≡ (263k)
```
  val wait : t -> unit
```

⟨*signature* Condition.set 263i⟩≡ (263k)
```
  val set : t -> unit
```

⟨*signature* Condition.free 263j⟩≡ (263k)
```
  val free : t -> unit
```

⟨commons/condition.mli 263k⟩≡
```

  (* abstract *)
  type t

  type condition_backend = {
    create: t -> unit;
    set: t -> unit;
    wait: t -> unit;
    free: t -> unit;
  }

  val backend: condition_backend ref

  ⟨signature Condition.create 263g⟩
  ⟨signature Condition.set 263i⟩
  ⟨signature Condition.wait 263h⟩
  ⟨signature Condition.free 263j⟩
```

F.1.2 core/condition.ml

⟨*type* Condition.t 264a⟩≡ (264b)
```
  type t = string
```

⟨commons/condition.ml 264b⟩≡
```
  (* Conditions *)
  open Common

  ⟨type Condition.t 264a⟩

  type condition_backend = {
    create: t -> unit;
    set: t -> unit;
    wait: t -> unit;
    free: t -> unit;
  }

  let default_backend () = {
    create = (fun _s -> ());
    set = (fun _s -> ());
    wait = (fun _s -> ());
    free = (fun _s -> ());
  }
  let backend = ref (default_backend ())

  let count = ref 0

  let create () =
    incr count;
    let var = spf "var%d" !count in
    (!backend).create var;
    var

  let set s =
    !backend.set s

  let wait s =
    !backend.wait s

  let free s =
    !backend.free s
```

F.1.3 misc/date.mli

⟨commons/date.mli 264c⟩≡
```

  ⟨signature Date.asc_wkday 261b⟩
  ⟨signature Date.asc_month 261c⟩
  ⟨signature Date.asc 261d⟩
  ⟨signature Date.asc_now 261e⟩

  ⟨signature Date.commonlog 261f⟩

  ⟨signature Date.compare_time 261g⟩
```

F.1.4 misc/date.ml

```
⟨commons/date.ml 265a⟩≡
  ⟨copyright header v6 14a⟩

  open Printf
  open Unix

  ⟨function Date.asc_wkday 262a⟩
  ⟨function Date.asc_month 262b⟩
  ⟨function Date.asc 262c⟩
  ⟨function Date.asc_now 262d⟩

  ⟨function Date.commonlog 262e⟩

  ⟨function Date.compare_time 262f⟩
```

F.1.5 misc/ebuffer.mli

```
⟨commons/ebuffer.mli 265b⟩≡

  (* Extensible buffers *)

  ⟨signature Ebuffer.t abstract 258h⟩

  ⟨signature Ebuffer.create 258a⟩

  ⟨signature Ebuffer.output_string 258b⟩
  ⟨signature Ebuffer.output_char 258c⟩
  ⟨signature Ebuffer.output 258d⟩

  ⟨signature Ebuffer.get 258e⟩
  ⟨signature Ebuffer.used 258f⟩

  ⟨signature Ebuffer.reset 258g⟩
```

F.1.6 misc/ebuffer.ml

```
⟨commons/ebuffer.ml 265c⟩≡
  ⟨copyright header v6 14a⟩

  ⟨type Ebuffer.t 258i⟩

  ⟨function Ebuffer.create 258j⟩

  ⟨function Ebuffer.reset 258k⟩

  ⟨function Ebuffer.newsize 258l⟩

  ⟨function Ebuffer.output_string 259a⟩
  ⟨function Ebuffer.output_char 259b⟩

  ⟨function Ebuffer.output 259c⟩

  ⟨function Ebuffer.get 259d⟩

  ⟨function Ebuffer.used 259e⟩
```

F.1.7 misc/error.mli

```
⟨commons/error.mli 266a⟩≡

  ⟨signature class Error.t 251e⟩

  ⟨signature Error.default 251f⟩

  ⟨signature Error.f 251a⟩
  ⟨signature Error.ok 251b⟩
  ⟨signature Error.choose 251c⟩
  ⟨signature Error.ari 251d⟩
```

F.1.8 misc/error.ml

```
⟨commons/error.ml 266b⟩≡
  open Common

  ⟨class Error.t 251i⟩

  class virtual t = object
   method virtual f : string -> unit
   method virtual ok : string -> unit
   method virtual choose : string -> bool
   method virtual ari : string -> int
  end

  class x = object
    inherit t
    method f _ = Logs.err (fun m -> m "TODO: Error.x.f")
    method ok _ = Logs.err (fun m -> m "TODO: Error.x.ok")
    method choose _ = failwith "TODO: Error.x.choose"
    method ari _ = failwith "TODO: Error.x.ari"
  end

  ⟨constant Error.default 251g⟩

  (* backward compatibility *)
  ⟨functions Error.xxx 251h⟩
```

F.1.9 misc/i18nprintf.mli

```
⟨signature I18nprintf.fprintf 266c⟩≡                                  (267d)
  val fprintf: out_channel -> ('a, out_channel, unit) format -> 'a
          (* [fprintf outchan format arg1 ... argN] formats the arguments
             [arg1] to [argN] according to the format string [format],
             and outputs the resulting string on the channel [outchan].

             The format is a character string which contains two types of
             objects:  plain  characters, which are simply copied to the
             output channel, and conversion specifications, each of which
             causes  conversion and printing of one argument.

             Conversion specifications consist in the [%] character, followed
             by optional flags and field widths, followed by one conversion
             character. The conversion characters and their meanings are:
  -          [d] or [i]: convert an integer argument to signed decimal
  -          [u]: convert an integer argument to unsigned decimal
  -          [x]: convert an integer argument to unsigned hexadecimal,
                 using lowercase letters.
  -          [X]: convert an integer argument to unsigned hexadecimal,
                 using uppercase letters.
  -          [s]: insert a string argument
  -          [c]: insert a character argument
  -          [f]: convert a floating-point argument to decimal notation,
                 in the style [dddd.ddd]
  -          [e] or [E]: convert a floating-point argument to decimal notation,
                 in the style [d.ddd e+-dd] (mantissa and exponent)
  -          [g] or [G]: convert a floating-point argument to decimal notation,
                 in style [f] or [e], [E] (whichever is more compact)
  -          [b]: convert a boolean argument to the string [true] or [false]
  -          [a]: user-defined printer. Takes two arguments and apply the first
                 one to [outchan] (the current output channel) and to the second
                 argument. The first argument must therefore have type
                 [out_channel -> 'b -> unit] and the second ['b].
                 The output produced by the function is therefore inserted
                 in the output of [fprintf] at the current point.
  -          [t]: same as [%a], but takes only one argument (with type
                 [out_channel -> unit]) and apply it to [outchan].
  -         Refer to the C library [printf] function for the meaning of
            flags and field width specifiers.

            If too few arguments are provided, printing stops just
            before converting the first missing argument. *)
```

⟨*signature* I18nprintf.printf 267a⟩≡ (267d)

```
  val printf: ('a, out_channel, unit) format -> 'a
          (* Same as [fprintf], but output on [stdout]. *)
```

⟨*signature* I18nprintf.eprintf 267b⟩≡ (267d)

```
  val eprintf: ('a, out_channel, unit) format -> 'a
          (* Same as [fprintf], but output on [stderr]. *)
```

⟨*signature* I18nprintf.sprintf 267c⟩≡ (267d)

```
  val sprintf: ('a, unit, string) format -> 'a
          (* Same as [printf], but return the result of formatting in a
             string. *)
```

⟨commons/i18nprintf.mli 267d⟩≡

```
  (***********************************************************************)
  (*                                                                     *)
  (*                           Objective Caml                            *)
  (*                                                                     *)
  (*            Xavier Leroy, projet Cristal, INRIA Rocquencourt         *)
  (*                                                                     *)
  (*  Copyright 1996 Institut National de Recherche en Informatique et   *)
  (*  Automatique.  Distributed only by permission.                      *)
  (*                                                                     *)
  (***********************************************************************)

  (* Module [Printf]: formatting printing functions *)

  ⟨signature I18nprintf.fprintf 266c⟩
  ⟨signature I18nprintf.printf 267a⟩
  ⟨signature I18nprintf.eprintf 267b⟩
  ⟨signature I18nprintf.sprintf 267c⟩
```

F.1.10 misc/i18nprintf.ml

⟨*function* I18nprintf.fprintf 268⟩≡ (270)

```
  let fprintf outchan format =
    let format = (Obj.magic format : string) in
    let outside_iso8859 = ref false in
    let rec doprn i =
      if i >= String.length format then
        Obj.magic ()
      else begin
        let c = String.unsafe_get format i in
        if c = '\027' then begin
      if i+2 < String.length format &&
         String.unsafe_get format (i+1) = '\040' &&
         String.unsafe_get format (i+2) = '\066' then
           outside_iso8859 := false
      else outside_iso8859 := true
        end;
        if c <> '%' || !outside_iso8859 then begin
          output_char outchan c;
          doprn (succ i)
        end else begin
          let j = skip_args (succ i) in
          match String.unsafe_get format j with
            '%' ->
              output_char outchan '%';
              doprn (succ j)
          | 's' ->
              Obj.magic(fun s ->
                if j <= i+1 then
                  output_string outchan s
                else begin
                  let p =
                    try
                      int_of_string (String.sub format (i+1) (j-i-1))
                    with _ ->
                      invalid_arg "I18nprintf.fprintf: bad %s format" in
                  if p > 0 && String.length s < p then begin
                    output_string outchan
                                  (String.make (p - String.length s) ' ');
                    output_string outchan s
                  end else if p < 0 && String.length s < -p then begin
                    output_string outchan s;
                    output_string outchan
                                  (String.make (-p - String.length s) ' ')
                  end else
                    output_string outchan s
                end;
                doprn (succ j))
          | 'c' ->
              Obj.magic(fun c ->
                output_char outchan c;
                doprn (succ j))
          | 'd' | 'o' | 'x' | 'X' | 'u' ->
              Obj.magic(fun n ->
                output_string outchan
                              (format_int (String.sub format i (j-i+1)) n);
                doprn (succ j))
          | 'f' | 'e' | 'E' | 'g' | 'G' ->
              Obj.magic(fun f ->
                output_string outchan
                              (format_float (String.sub format i (j-i+1)) f);
                doprn (succ j))
          | 'b' ->
              Obj.magic(fun b ->
                output_string outchan (string_of_bool b);
                doprn (succ j))
          | 'a' ->
              Obj.magic(fun printer arg ->
                printer outchan arg;
                doprn(succ j))
          | 't' ->
              Obj.magic(fun printer ->
                printer outchan;
                doprn(succ j))
          | _c ->
              invalid_arg ("I18nprintf.fprintf: unknown format")
        end
      end

    and skip_args j =
      match String.unsafe_get format j with
        '0' .. '9' | ' ' | '.' | '-' -> skip_args (succ j)
      | _c -> j

    in doprn 0
```

⟨*function* I18nprintf.sprintf 269⟩≡ (270)

```
 let sprintf format =
   let format = (Obj.magic format : string) in
   let outside_iso8859 = ref false in
   let rec doprn start i accu =
     if i >= String.length format then begin
       let res =
         if i > start
         then String.sub format start (i-start) :: accu
         else accu in
       Obj.magic(String.concat "" (List.rev res))
     end else
       let c = String.unsafe_get format i in
       if c = '\027' then begin
     if i+2 < String.length format &&
        String.unsafe_get format (i+1) = '\040' &&
        String.unsafe_get format (i+2) = '\066' then
          outside_iso8859 := false
     else outside_iso8859 := true
       end;
       if c <> '%' || !outside_iso8859 then
         doprn start (i+1) accu
       else begin
         let accu1 =
           if i > start then
           String.sub format start (i-start) :: accu
           else accu in
         let j = skip_args (succ i) in
         match String.unsafe_get format j with
           '%' ->
             doprn j (succ j) accu1
         | 's' ->
             Obj.magic(fun s ->
               let accu2 =
                 if j <= i+1 then
                   s :: accu1
                 else begin
                   let p =
                     try
                       int_of_string (String.sub format (i+1) (j-i-1))
                     with _ ->
                       invalid_arg "I18nprintf.fprintf: bad %s format" in
                   if p > 0 && String.length s < p then
                     s :: String.make (p - String.length s) ' ' :: accu1
                   else if p < 0 && String.length s < -p then
                     String.make (-p - String.length s) ' ' :: s :: accu1
                   else
                     s :: accu1
                 end in
               doprn (succ j) (succ j) accu2)
         | 'c' ->
             Obj.magic(fun c ->
               doprn (succ j) (succ j) (String.make 1 c :: accu1))
         | 'd' | 'o' | 'x' | 'X' | 'u' ->
             Obj.magic(fun n ->
               doprn (succ j) (succ j)
                     (format_int (String.sub format i (j-i+1)) n :: accu1))
         | 'f' | 'e' | 'E' | 'g' | 'G' ->
             Obj.magic(fun f ->
               doprn (succ j) (succ j)
                     (format_float (String.sub format i (j-i+1)) f :: accu1))
         | 'b' ->
             Obj.magic(fun b ->
               doprn (succ j) (succ j) (string_of_bool b :: accu1))
         | 'a' ->
             Obj.magic(fun printer arg ->
               doprn (succ j) (succ j) (printer () arg :: accu1))
         | 't' ->
             Obj.magic(fun printer ->
               doprn (succ j) (succ j) (printer () :: accu1))
         | _c ->
             invalid_arg ("I18nprintf.sprintf: unknown format")
       end

   and skip_args j =
     match String.unsafe_get format j with
         '0' .. '9' | ' ' | '.' | '-' -> skip_args (succ j)
     | _c -> j

   in doprn 0 0 []
```

⟨commons/i18nprintf.ml 270⟩≡

```
(***********************************************************************)
(*                                                                     *)
(*                           Objective Caml                            *)
(*                                                                     *)
(*            Xavier Leroy, projet Cristal, INRIA Rocquencourt         *)
(*                                                                     *)
(*  Copyright 1996 Institut National de Recherche en Informatique et   *)
(*  Automatique.  Distributed only by permission.                      *)
(*                                                                     *)
(***********************************************************************)
```

```
external format_int: string -> int -> string = "caml_format_int"
external format_float: string -> float -> string = "caml_format_float"

⟨function I18nprintf.fprintf 268⟩

let printf fmt = fprintf stdout fmt
and eprintf fmt = fprintf stderr fmt

⟨function I18nprintf.sprintf 269⟩
```

F.1.11 misc/ibtree.mli

⟨commons/ibtree.mli 271a⟩≡
```
module type S =
  sig
    type key
    type 'a t
    val empty: 'a t
    val add: (key * key) -> 'a -> 'a t -> 'a t
    val find: key -> 'a t -> 'a

    val find_interval : key -> 'a t -> key * key
  end

module Make(Ord: Map.OrderedType): (S with type key = Ord.t)
```

F.1.12 misc/ibtree.ml

⟨commons/ibtree.ml 271b⟩≡
```
(* Simple binary trees with no redondant elements, and no delete function *)

module type S =
  sig
    type key
    type 'a t
    val empty: 'a t
    val add: (key * key) -> 'a -> 'a t -> 'a t
    val find: key -> 'a t -> 'a
    val find_interval : key -> 'a t -> key * key
  end


module Make(Ord: Map.OrderedType) = struct

type key = Ord.t

type balance = Eq | Le | Ri

type 'a element = {
    interval : key * key;
    image : 'a
  }

type 'a t =
 | Empty
 | Node of 'a node

and 'a node =
 {balance : balance; height : int;
  left : 'a t; element : 'a element; right : 'a t}

let height = function
  | Empty -> 0
  | Node {height = h; _} -> h

let create_node l e r =
  let hl = height l in
  let hr = height r in
  Node
   {
     balance =
       (if hl = hr
       then Eq
       else
         if hl < hr then Ri else Le
        )
     ;
     height = 1 + (if hr > hl then hr else hl);
     left = l;
     element = e;
     right = r;
   }

let turn_right = function
 | Empty -> Empty
 | Node
    {left =
      Node
       {balance = Eq | Ri;
        left = lle; element = le;
        right = Node {left = lrle; element = rle; right = rrle; _}; _};
     element = e; right = re; _} ->
      create_node (create_node lle le lrle) rle (create_node rrle e re)
 | Node
    {left =
      Node
       {balance = Le | Eq; left = lle; element = le; right = rle; _};
     element = e;
     right = re; _} -> create_node lle le (create_node rle e re)
 | _ -> failwith "turn_right"

and turn_left = function
 | Empty -> Empty
 | Node
    {left = le; element = e;
     right =
      Node
       {balance = Eq | Le;
        left = Node {left = llre; element = lre; right = rlre; _};
        element = re; right = rre; _}; _} ->
     create_node (create_node le e llre) lre (create_node rlre re rre)
 | Node
    {left = le; element = e;
     right =
      Node
       {balance = Ri | Eq; left = lre; element = re; right = rre; _}; _} ->
     create_node (create_node le e lre) re rre
 | _ -> failwith "turn_left"
```

```
let reball hr l e nr =
 let hnr = height nr and nt = create_node l e nr in
 if hnr > hr then turn_left nt else nt

let rebalr hl nl e r =
 let hnl = height nl and nt = create_node nl e r in
 if hnl > hl then turn_right nt else nt

(* Si les intervalles sont disjoints, la comparaison des bornes inferieures
   est un ordre *)

let compare_elements x y =
  Ord.compare (fst x.interval) (fst y.interval)

let rec insert x t =
 match t with
 | Empty ->
    Node {balance = Eq; height = 1; left = Empty; element = x; right = Empty}
 | Node {balance = b; left = l; element = e; right = r; _} ->
    let c = compare_elements x e in
    if c = 0 then t else
    if c > 0 then
     if b = Ri then reball (height r) l e (insert x r)
      else create_node l e (insert x r) else
    if b = Le then rebalr (height l) (insert x l) e r
     else create_node (insert x l) e r


let empty = Empty

let rec find pos = function
| Empty -> raise Not_found
| Node {left = l; element = e ; right = r; _} ->
   let c_start = Ord.compare pos (fst e.interval) in
   if c_start < 0 then find pos l else
   let c_stop = Ord.compare pos (snd e.interval) in
   if c_stop < 0 then e.image else find pos r


let rec find_interval pos = function
| Empty -> raise Not_found
| Node {left = l; element = e ; right = r; _} ->
   let c_start = Ord.compare pos (fst e.interval) in
   if c_start < 0 then find_interval pos l else
   let c_stop = Ord.compare pos (snd e.interval) in
   if c_stop < 0 then e.interval else find_interval pos r


let add i v = insert {interval = i; image = v}


(* Application to anchors
type position = int * int

type anchor_position = {start : position; stop : position}

let rec anchor_of_pos pos = function
| Empty -> raise Not_found
| Node {left = l; element = e; right = r} ->
```

```
    let c_start = compare pos e.start in
    if c_start < 0 then anchor_of_pos pos l else
    let c_stop = compare pos e.stop in
    if c_stop < 0 then e else anchor_of_pos pos r

 let anchors_table = ref Empty

 let add_anchor a = anchors_table := insert a !anchors_table
 let find_anchor mouse_pos = anchor_of_pos mouse_pos !anchors_table

  *)
 (* Exemples
 let anchors = [
  {start = 1,3; stop = 1,7};
  {start = 1,13; stop = 1,17};
  {start = 1,30; stop = 1,70};
  {start = 10,3; stop = 10,7};
  {start = 11,3; stop = 12,1};
  {start = 12,3; stop = 13,17};
  {start = 14,30; stop = 15,7};
  {start = 100,3; stop = 100,7};
  {start = 101,3; stop = 101,7};
  {start = 101,31; stop = 101,37}
  ]
 do_list add_anchor anchors

 find_anchor (1,6)
 find_anchor (12,0)
 find_anchor (11,7)
 find_anchor (101,3)
 find_anchor (15,3)
 find_anchor (16,3)

 (* A bit long (more than 3 seconds in Caml Light) *)
 for i = 0 to 10000 do
  add_anchor {start = i, 1+i/2; stop = i, i}
 done
 *)
 end
```

F.1.13 i18n/lang.mli

⟨commons/lang.mli 274a⟩≡
  ⟨*signature* Lang.lang 224i⟩

F.1.14 i18n/lang.ml

⟨commons/lang.ml 274b⟩≡

  ⟨*function* Lang.lang 224j⟩

```
 (* detect and set LANG information *)
```

F.1.15 misc/log.mli

⟨commons/log.mli 274c⟩≡
  ⟨*signature* Log.debug_mode 243a⟩

```
  ⟨signature Log.f 243c⟩
  ⟨signature Log.debug 243e⟩
```

F.1.16 misc/log.ml

```
⟨commons/log.ml 275a⟩≡

  ⟨constant Log.debug_mode 243b⟩

  ⟨function Log.f 243d⟩

  ⟨function Log.debug 243f⟩
```

F.1.17 core/low.mli

```
⟨signature Low.read 275b⟩≡                                                    (275h)
  val count_read : (bytes -> int -> int -> int) -> bytes -> int -> int -> int
    (* wraps any read function with tachymeter byte accounting *)

  val read : Unix.file_descr -> bytes -> int -> int -> int
    (* Unix.read wrapper, to be used when data transferred has to
       be counted by the tachymeter
     *)

⟨signature Low.fork 275c⟩≡                                                    (275h)
  val fork : unit -> int
    (* Unix.fork wrapper. Catches zombies *)

⟨signature Low.add_fileinput 275d⟩≡                                           (275h)
  val add_fileinput : Unix.file_descr -> (unit -> unit) -> unit

⟨signature Low.remove_fileinput 275e⟩≡                                        (275h)
  val remove_fileinput: Unix.file_descr -> unit
    (* Wrapping of Tk fileinput functions, with feedback on the tachymeter *)

⟨signature Low.busy 275f⟩≡                                                    (275h)
  val busy : ('a -> 'b) -> 'a -> 'b
    (* Busy feedback during this application *)

⟨signature Low.update_idletasks 275g⟩≡                                        (275h)
  val update_idletasks : unit -> unit

⟨commons/low.mli 275h⟩≡

  ⟨signature Low.read 275b⟩

  val read_line: Unix.file_descr -> string
  val read_line_fn : (bytes -> int -> int -> int) -> string

  ⟨signature Low.fork 275c⟩

  ⟨signature Low.add_fileinput 275d⟩
  ⟨signature Low.remove_fileinput 275e⟩

  ⟨signature Low.busy 275f⟩

  ⟨signature Low.global_time 196d⟩
```

```
⟨signature Low.add_task 196a⟩

class  virtual tachymeter : object
  method virtual report_cnx : int -> unit     (* displays number of active cnx *)
  method virtual report_busy : bool -> unit   (* displays busy status *)
  method virtual report_traffic : int -> int -> int -> unit
       (* [report_traffic tick_duration total sample] displays traffic
          from [total] and [sample] in last [tick_duration] *)
  method virtual quit : unit
end
⟨signature Low.cur_tachy 196j⟩

⟨signature Low.init 196f⟩

val update_idletasks_backend: (unit -> unit) ref
⟨signature Low.update_idletasks 275g⟩
```

F.1.18 core/low.ml

⟨*constant* Low.cur_tachy 276a⟩≡ (277g)

```
  let cur_tachy = ref (new no_tachy :> tachymeter)
```

⟨*function* Low.read 276b⟩≡ (277g)

```
  let count_read read_fn buf offs l =
    let n = read_fn buf offs l in
    bytes_read := !bytes_read + n;
    sample_read := !sample_read + n;
    n

  let read fd = count_read (Unix.read fd)
```

⟨*function* Low.add_fileinput 276c⟩≡ (277g)

```
  let add_fileinput fd f =
    incr pending_read;
    !cur_tachy#report_cnx !pending_read;
    Fileevent_.add_fileinput fd f
```

⟨*function* Low.remove_fileinput 276d⟩≡ (277g)

```
  let remove_fileinput fd =
    decr pending_read;
    !cur_tachy#report_cnx !pending_read;
    Fileevent_.remove_fileinput fd
```

⟨*function* Low.fork 276e⟩≡ (277g)

```
  let fork () =
   begin try
    while
      let p, _s = Unix.waitpid [Unix.WNOHANG] 0 in
        (*
        Printf.eprintf "%d\n" p;
        begin match s with
          WEXITED n -> Printf.eprintf "Exit %d\n" n
         | WSIGNALED(n,_) ->
             Printf.eprintf "SIG %d\n" n
         | WSTOPPED n -> Printf.eprintf "Stopped %d\n" n
        end;
        flush Pervasives.stderr;
        *)
        p <> 0
   do () done
   with
    Unix.Unix_error(_,_,_) -> ()
   end;
   (* Don't let children play stupid games *)
   match Unix.fork() with
     0 -> at_exit (fun () -> sys_exit 0); 0
   | n -> n
```

⟨*function* Low.busy 277a⟩≡ (277g)

```
let busy f x =
  !cur_tachy#report_busy true;
  try
    let v = f x in
    !cur_tachy#report_busy false; v
  with
    e ->
      !cur_tachy#report_busy false;
      raise e
```

⟨*constant* Low.tick_duration 277b⟩≡ (277g)

```
let tick_duration = 500
```

⟨*function* Low.add_task 277c⟩≡ (277g)

```
let add_task f = tasks := f :: !tasks
```

⟨*constant* Low.last_update 277d⟩≡ (277g)

```
(* We need manual refresh for progressive display (?), but we don't
   want to do it too frequently *)
let last_update = ref !global_time
```

⟨*function* Low.update_idletasks 277e⟩≡ (277g)

```
let update_idletasks () =
  if !global_time <> !last_update then begin
    !update_idletasks_backend ();
    last_update := !global_time
  end
```

⟨*global* Low.pending_read 277f⟩≡ (277g)

```
let pending_read = ref 0
```

⟨commons/low.ml 277g⟩≡

```
(* Wrapping of some low-level functions *)

(* Tachymeter support *)
class  virtual tachymeter = object
  method virtual report_cnx : int -> unit     (* displays number of active cnx *)
  method virtual report_busy : bool -> unit   (* displays busy status *)
  method virtual report_traffic : int -> int -> int -> unit
       (* [report_traffic tick_duration total sample] displays traffic
          from [total] and [sample] in last [tick_duration] *)
  method virtual quit : unit
  end

class no_tachy = object
  inherit tachymeter

  method report_cnx _cnx = ()
  method report_busy _flag = ()
  method report_traffic _tick _total _sample = ()
  method quit = ()
end

⟨constant Low.cur_tachy 276a⟩

(* for the tachymeter *)
⟨global Low.bytes_read 196h⟩
⟨global Low.sample_read 196i⟩

⟨function Low.read 276b⟩

(*
 * Read a line (terminated by \n or \r\n).
 *   strips terminator !
 *)
let read_line_fn (read_fn : bytes -> int -> int -> int) =
  let rec read_rec (buf : bytes) bufsize offs =
    let n = count_read read_fn buf offs 1 in
      if n = 0 then raise End_of_file
      else if Bytes.get buf offs = '\n'
           then (* strips \n and possibly \r  *)
             let len = if offs >= 1 && Bytes.get buf (offs-1) = '\r' then offs-1
                       else offs in
               Bytes.sub_string buf 0 len
           else let offs = succ offs in
                  if offs = bufsize
                  then read_rec (Bytes.cat buf (Bytes.create 128)) (bufsize + 128) offs
                  else read_rec buf bufsize offs in
  read_rec (Bytes.create 128) 128 0

let read_line fd = read_line_fn (read fd)


⟨global Low.pending_read 277f⟩
let _action = ref (fun _ -> ())

⟨function Low.add_fileinput 276c⟩

⟨function Low.remove_fileinput 276d⟩

(* We catch dead children here, to avoid large number of zombies.
   I know about SICHLD of course, but I hate interrupted syscalls
 *)

external sys_exit : int -> 'a = "caml_sys_exit"

⟨function Low.fork 276e⟩


⟨function Low.busy 277a⟩

⟨constant Low.global_time 196e⟩
⟨constant Low.tick_duration 277b⟩

⟨constant Low.tasks 196b⟩

⟨function Low.refresh 196c⟩

⟨function Low.add_task 277c⟩
```

```
  ⟨function Low.init 196g⟩

  let update_idletasks_backend =
    ref (fun _ -> failwith "no update_idletasks defined")

  ⟨constant Low.last_update 277d⟩
  ⟨function Low.update_idletasks 277e⟩
```

F.1.19 misc/mlist.mli

```
⟨commons/mlist.mli 279a⟩≡
  ⟨signature Mlist.hdn 254a⟩
  ⟨signature Mlist.tln 254b⟩

  ⟨signature Mlist.except_assoc 254c⟩
  ⟨signature Mlist.exceptq 254d⟩
  ⟨signature Mlist.rev_do_list 254e⟩

  ⟨signature Mlist.do_listi 254f⟩
```

F.1.20 misc/mlist.ml

```
⟨commons/mlist.ml 279b⟩≡
  (*
   * List utilities
   *)

  ⟨function Mlist.tln 254g⟩
  ⟨function Mlist.hdn 254h⟩

  ⟨function Mlist.except_assoc 254i⟩
  ⟨function Mlist.exceptq 255a⟩

  ⟨function Mlist.rev_do_list 255b⟩

  ⟨function Mlist.do_listi 255c⟩
```

F.1.21 misc/mstring.mli

```
⟨commons/mstring.mli 279c⟩≡

  ⟨signature Mstring.split_str 255d⟩
  ⟨signature Mstring.get_suffix 255e⟩

  ⟨signature Mstring.hex_to_dec 255f⟩
  ⟨signature Mstring.dec_to_hex 255g⟩

  ⟨signature Mstring.hex_to_string 255h⟩

  ⟨signature Mstring.gensym 255i⟩
  ⟨signature Mstring.egensym 255j⟩

  ⟨signature Mstring.rem_trailing_sp 255k⟩

  val utf8_length : string -> int
```

⟨*signature* Mstring.catenate_sep 255l⟩

⟨*signature* Mstring.norm_crlf 255m⟩

F.1.22 misc/mstring.ml

⟨commons/mstring.ml 280a⟩≡
```
  (*
   * String utilities
   *)
```

⟨*function* Mstring.split_str 256a⟩

⟨*function* Mstring.get_suffix 256b⟩

⟨*function* Mstring.hex_to_dec 256c⟩
⟨*function* Mstring.dec_to_hex 256d⟩
⟨*function* Mstring.hex_to_string 256e⟩

⟨*constant* Mstring.gensym 256f⟩
⟨*function* Mstring.egensym 257a⟩

⟨*function* Mstring.rem_trailing_sp 257b⟩

```
  (* Count Unicode codepoints in a UTF-8 encoded string.
   * Unlike String.length (bytes) this matches Tk's CharOffset counting. *)
  let utf8_length s =
    let n = String.length s in
    let count = ref 0 in
    let i = ref 0 in
    while !i < n do
      let c = Char.code (String.unsafe_get s !i) in
      if c land 0x80 = 0 then (incr count; incr i)
      else if c land 0xE0 = 0xC0 then (incr count; i := !i + 2)
      else if c land 0xF0 = 0xE0 then (incr count; i := !i + 3)
      else if c land 0xF8 = 0xF0 then (incr count; i := !i + 4)
      else incr i (* continuation byte: skip *)
    done;
    !count
```

⟨*function* Mstring.catenate_sep 257c⟩

⟨*function* Mstring.norm_crlf 257d⟩

F.1.23 misc/msys.mli

⟨commons/msys.mli 280b⟩≡

⟨*signature* Msys.tilde_subst 259f⟩

⟨*signature* Msys.rm 259g⟩

⟨*signature* Msys.fsize 259h⟩

⟨*signature* Msys.mktemp 260a⟩

F.1.24 misc/msys.ml

```
⟨commons/msys.ml 281a⟩≡
  ⟨copyright header v6 14a⟩

  open Unix

  (* Tilde substitution *)

  ⟨function Msys.next_slash 260b⟩

  ⟨function Msys.tilde_subst 260c⟩

  ⟨function Msys.rm 260d⟩
  ⟨function Msys.rmdir 260e⟩

  ⟨function Msys.fsize 260f⟩

  ⟨constant Msys.tmp_dir 260g⟩

  ⟨constant Msys.mktemp 261a⟩
```

F.1.25 misc/i18n.mli

```
⟨commons/i18n.mli 281b⟩≡
  ⟨signature I18n.message_file 224f⟩
  ⟨signature I18n.language 224c⟩

  ⟨signature I18n.sprintf 224k⟩

  (* pad's compact alias *)
  val s_: ('a, unit, string) format -> 'a

  ⟨signature I18n.menu_option 224l⟩
  ⟨signature I18n.menu_pattern 224m⟩

  val translate: string -> string
```

F.1.26 misc/i18n.ml

```
⟨commons/i18n.ml 281c⟩≡

  ⟨function I18n.fprintf 225a⟩
  ⟨function I18n.sprintf 225b⟩

  ⟨constant I18n.language 224d⟩
  ⟨constant I18n.message_file 224g⟩

  ⟨function I18n.read_transl_file 225c⟩

  ⟨type I18n.translation_table 225d⟩

  ⟨constant I18n.transl_table 226a⟩

  ⟨function I18n.translate 226b⟩

  ⟨function I18n.fprintf (./commons/i18n.ml) 226c⟩
```

```
⟨function I18n.sprintf (./commons/i18n.ml) 226d⟩

let s_ fmt = sprintf fmt

⟨function I18n.menu_option 226e⟩

⟨exception I18n.Found 226f⟩

⟨function I18n.menu_pattern 226g⟩
```

F.1.27 misc/munix.ml

```
⟨function Munix.execvp 282a⟩≡                                          (284a)
  (* If execvp fails in one of our children, it may be dangerous to leave
     the program running, since we don't know how Tk would react *)
  let execvp s args =
    try
      Unix.execvp s args
    with
      Unix_error(e, _, _) ->
         Printf.eprintf "%s\n" (Unix.error_message e);
         flush Stdlib.stderr;
         (* nosemgrep: do-not-use-exit *)
         exit 1

⟨constant Munix.quote 282b⟩≡                                           (284a)
  let quote = Str.regexp "'"

⟨function Munix.quote_for_shell 282c⟩≡                                 (284a)
  let quote_for_shell s =
    sprintf "'%s'" (Str.global_replace quote "'\\''" s)

⟨function Munix.system 282d⟩≡                                          (284a)
  (* Wrapping of Sys.command with trivial arg quoting *)
  let system cmd args back =
    let b = Ebuffer.create 128 in
     Ebuffer.output_string b cmd;
     List.iter (fun s ->
       Ebuffer.output_char b ' ';
       Ebuffer.output_string b (quote_for_shell s))
      args;
     if back then Ebuffer.output_string b " &";
     Sys.command (Ebuffer.get b)

⟨function Munix.eval_cmd 282e⟩≡                                        (284a)
  let eval_cmd cmd args back =
   let _ = system cmd args back in ()

⟨function Munix.write_string 282f⟩≡                                    (284a)
  let write_string fd (s : string) =
    ignore (write fd (Bytes.of_string s) 0 (String.length s))

⟨function Munix.read_line 282g⟩≡                                       (284a)
```

⟨*function* Munix.full_random_init 283a⟩≡ (284a)
```
  let full_random_init () =
    try
      let env = environment () in
      let vect =
        Array.append (Array.map Hashtbl.hash env)
             [| getpid(); Stdlib.truncate (time()); (* JPF: bogus *)
                getuid(); getgid();
                Hashtbl.hash (getlogin()) |] in
      Random.full_init vect
    with
      _ -> ()
```

⟨*function* Munix.digdir 283b⟩≡ (284a)
```
  let rec digdir dir perm =
    (* try to create the directory dir *)
    if Sys.file_exists dir then ()
    else begin
      let pdir = Filename.dirname dir in
      digdir pdir perm;
      Unix.mkdir dir perm
    end
```

⟨*constant* Munix.dns 283c⟩≡ (284a)
```
  (* DNS Caching. It really helps on slow lines... *)
  let dns = Hashtbl.create 307
```

⟨*function* Munix.gethostbyname 283d⟩≡ (284a)
```
  let gethostbyname h =
    try Hashtbl.find dns h
    with
      Not_found ->
        let addr = Unix.gethostbyname h in
          Hashtbl.add dns h addr;
       addr
```

⟨*toplevel* Munix._1 283e⟩≡ (284a)
```
  let _ =
    full_random_init()
```

⟨*constant* Munix.vars 283f⟩≡ (284a)
```
  (* Hack to run some external command with parameter substitution
   * The command is a string containing $X
   * The arguments are [X, v]
   * For arguments not substituted, add them at the end,
   *)
  let vars = Str.regexp "\\$[A-Z]+"
```

⟨*function* Munix.system_eval 283g⟩≡ (284a)
```
  let system_eval cmd args back =
    let replaced = ref []
    and _qargs = List.map (fun (x, v) -> x, quote_for_shell v) args
    in
    let replfun s =
      let matched = Str.matched_string s in
      let thevar = String.sub matched 1 (String.length matched - 1) in
      try
        let res = List.assoc thevar args in
        replaced := thevar :: !replaced;
        res
      with
        Not_found -> matched
    in
    (* replace vars *)
    let scmd = Str.global_substitute vars replfun cmd in
    (* for vars that haven't been replaced, add them at the end
     * (backward compatibility with our previous versions)
     *)
    let remaining = ref [] in
    List.iter (fun (x,v) ->
      if not (List.mem x !replaced) then remaining := v :: !remaining)
      args;
    system scmd (List.rev !remaining) back
```

⟨commons/munix.ml 284a⟩≡

```
  open Printf
  open Unix

  (*
   * Simple Unix utilities
   *)


  ⟨function Munix.execvp 282a⟩

  ⟨constant Munix.quote 282b⟩
  ⟨function Munix.quote_for_shell 282c⟩

  ⟨function Munix.system 282d⟩

  ⟨function Munix.eval_cmd 282e⟩

  ⟨function Munix.write_string 282f⟩

  ⟨function Munix.read_line 282g⟩


  ⟨function Munix.full_random_init 283a⟩

  ⟨function Munix.digdir 283b⟩

  ⟨constant Munix.dns 283c⟩
  ⟨function Munix.gethostbyname 283d⟩

  ⟨toplevel Munix._1 283e⟩

  ⟨constant Munix.vars 283f⟩

  ⟨function Munix.system_eval 283g⟩
```

F.1.28 misc/feed.mli

⟨commons/feed.mli 284b⟩≡

```

  ⟨type Feed.internal 93d⟩

  ⟨type Feed.t 93e⟩

  ⟨signature Feed.of_fd 93f⟩
  ⟨signature Feed.internal 95a⟩
```

F.1.29 misc/feed.ml

```
⟨commons/feed.ml 285a⟩≡

  ⟨type Feed.internal 93d⟩

  ⟨type Feed.t 93e⟩

  ⟨function Feed.of_fd 93g⟩

  ⟨function Feed.internal 95b⟩
```

F.1.30 tk/glevents.mli

```
⟨commons/glevents.mli 285b⟩≡
  open Tk

  ⟨signature Glevents.get 49b⟩
  ⟨signature Glevents.reset 49c⟩
```

F.1.31 tk/glevents.ml

```
⟨commons/glevents.ml 285c⟩≡
  open Printf
  open Tk

  ⟨constant Glevents.events 49d⟩

  ⟨constant Glevents.builtin_defaults 50a⟩

  ⟨constant Glevents.get 50b⟩

  ⟨function Glevents.reset 50c⟩
```

F.1.32 misc/hotlist.ml

```
⟨commons/hotlist.ml 285d⟩≡
  open I18n
  ⟨constant Hotlist.program 206c⟩

  ⟨function Hotlist.f 206d⟩
```

F.1.33 globals/version.mli

```
⟨globals/version.mli 285e⟩≡

  ⟨signature Version.number 15d⟩
  ⟨signature Version.http 15b⟩
  ⟨signature Version.about 15f⟩

  ⟨signature Version.initurl 32b⟩
  ⟨signature Version.html 32d⟩
  ⟨signature Version.helpurl 209g⟩
  ⟨signature Version.home 47e⟩
```

F.1.34 globals/version.ml

```
⟨globals/version.ml 286a⟩≡
  (* To merge FR and JP strings correctly, you have to encode the characters
   * more than 0x7F, for example "Fran\231ois".
   *)

  ⟨constant Version.number 15e⟩
  ⟨constant Version.version_number 16⟩

  ⟨constant Version.http 15c⟩

  ⟨function Version.about 15g⟩

  ⟨function Version.home 47f⟩

  ⟨function Version.initurl 32c⟩

  ⟨function Version.helpurl 209h⟩

  ⟨function Version.html 32e⟩
```

F.2 www/

F.2.1 www/uri.mli

```
⟨www/uri.mli 286b⟩≡

  ⟨type Uri.abs_uri 17c⟩

  ⟨signature Uri.is_absolute 17d⟩
```

F.2.2 www/uri.ml

```
⟨www/uri.ml 286c⟩≡

  ⟨type Uri.abs_uri 17c⟩

  ⟨function Uri.is_absolute 18a⟩
```

F.2.3 www/url.mli

```
⟨www/url.mli 286d⟩≡

  ⟨type Url.protocol 17b⟩

  ⟨signature Url.string_of_protocol 247b⟩

  ⟨type Url.t 17a⟩

  ⟨signature Url.string_of 247d⟩
  ⟨signature Url.distant_path 103a⟩

  ⟨exception Url.Url_Lexing 252b⟩
```

F.2.4 www/url.ml

```
⟨www/url.ml 287a⟩≡

  ⟨type Url.protocol 17b⟩
  ⟨function Url.string_of_protocol 247c⟩

  ⟨type Url.t 17a⟩

  ⟨exception Url.Url_Lexing 252b⟩
  ⟨exception Url.Invalid_url 252c⟩

  ⟨function Url.string_of 247e⟩

  ⟨function Url.distant_path 103b⟩
```

F.2.5 www/urlenc.mli

```
⟨www/urlenc.mli 287b⟩≡

  ⟨signature Urlenc.decode 55b⟩
  ⟨signature Urlenc.encode 55d⟩

  ⟨signature Urlenc.strict_form_standard 57a⟩
  ⟨signature Urlenc.form_encode 56e⟩
  ⟨signature Urlenc.form_decode 56c⟩

  ⟨signature Urlenc.unquote 54c⟩
```

F.2.6 www/urlenc.ml

```
⟨www/urlenc.ml 287c⟩≡
  ⟨copyright header v6 14a⟩

  open Mstring

  ⟨function Urlenc.hexchar 56b⟩

  ⟨function Urlenc.decode 55c⟩

  ⟨constant Urlenc.keep_quoted 55a⟩
  ⟨function Urlenc.unquote 54d⟩

  ⟨function Urlenc.encode 56a⟩


  ⟨constant Urlenc.strict_form_standard 57b⟩

  ⟨function Urlenc.form_encode 56f⟩
  ⟨constant Urlenc.form_decode 56d⟩
```

F.2.7 www/lexurl.mli

```
⟨www/lexurl.mli 288a⟩≡

  ⟨signature Lexurl.f 51a⟩
  ⟨signature Lexurl.make 51b⟩
  ⟨signature Lexurl.maken 53e⟩

  ⟨signature Lexurl.remove_dots 53g⟩
  ⟨signature Lexurl.normalize 53c⟩
```

F.2.8 www/lexurl.mll

```
⟨www/lexurl.mll 288b⟩≡
  {
  open Common
  open Mlist
  open Url

  ⟨function Lexurl.normalize_port 52b⟩
  ⟨function Lexurl.normalize_host 52d⟩
  }

  ⟨function Lexurl.f 51d⟩

  ⟨function Lexurl.slashslash 52a⟩

  ⟨function Lexurl.userpass 215i⟩

  ⟨function Lexurl.hostport 52c⟩

  ⟨function Lexurl.pathsearch 53a⟩

  ⟨functions Lexurl.xxx 53b⟩

  {

  ⟨function Lexurl.make 51c⟩

  ⟨function Lexurl.remove_dots 54a⟩

  ⟨function Lexurl.maken 53f⟩

  ⟨function Lexurl.normalize 53d⟩

  }
```

F.2.9 www/hyper.mli

```
⟨www/hyper.mli 288c⟩≡
  (* An hypertext(media) link on the Web *)

  ⟨type Hyper.link_method 18c⟩

  ⟨signature Hyper.parse_method 58b⟩

  ⟨type Hyper.link 18b⟩
  ⟨signature Hyper.default_link 18d⟩

  ⟨type Hyper.link_error 252e⟩

  ⟨exception Hyper.Invalid_link 252d⟩

  ⟨signature Hyper.urlconcat 57e⟩

  ⟨signature Hyper.resolve 57c⟩
  ⟨signature Hyper.string_of 249a⟩
```

F.2.10 www/hyper.ml

```
⟨www/hyper.ml 289a⟩≡
  open I18n
  open Printf

  open Uri
  open Url

  (* An hypertext(media) link on the Web *)

  ⟨type Hyper.link_method 18c⟩

  ⟨function Hyper.parse_method 58c⟩


  ⟨type Hyper.link 18b⟩

  ⟨type Hyper.link_error 252e⟩

  ⟨function Hyper.default_link 18e⟩

  ⟨exception Hyper.Invalid_link 252d⟩

  ⟨function Hyper.urlconcat 58a⟩

  ⟨function Hyper.resolve 57d⟩

  ⟨function Hyper.string_of 249b⟩
```

F.2.11 www/maps.mli

```
⟨www/maps.mli 289b⟩≡
  ⟨type Maps.area_kind 154a⟩

  ⟨type Maps.area 154b⟩

  ⟨type Maps.map 154c⟩

  ⟨type Maps.t 154d⟩

  ⟨type Maps.map_status 154e⟩

  ⟨signature Maps.parse_coords 154f⟩
  ⟨signature Maps.get 154g⟩

  val broadcast_backend: (string -> unit) ref
  ⟨signature Maps.add 154h⟩
```

F.2.12 www/maps.ml

```
⟨www/maps.ml 290a⟩≡
  open Printf

  (* Client-side image maps:
       the "only" difficulty in implementing client-side image maps is that
       the map may well come *after* the image in the document. In general,
       anyway, the map may be an arbitrary URL.

     We thus have to implement a general delay mechanism for maps : the idea
     here is to use a table of maps, each map being accessed by an URI (that is,
     an URL plus a fragment).

     PROBLEM: we have no idea in general when to flush this table.

   *)

  ⟨type Maps.area_kind 154a⟩

  ⟨type Maps.area 154b⟩

  ⟨type Maps.map 154c⟩

  ⟨type Maps.t 154d⟩


  ⟨type Maps.map_status 154e⟩

  ⟨constant Maps.table 154i⟩

  ⟨constant Maps.coord_sep 154j⟩
  ⟨function Maps.parse_coords 154k⟩

  let broadcast_backend = ref (fun _ev -> failwith "no broadcast defined")

  ⟨function Maps.add 155a⟩

  ⟨function Maps.get 155b⟩
```

F.2.13 www/www.mli

```
⟨www/www.mli 290b⟩≡
  ⟨type Www.request 18f⟩

  ⟨exception Www.Invalid_request 252f⟩

  ⟨signature Www.make 19b⟩

  (* Table of unresolved active connexions *)
  module UrlSet : Set.S with type elt = Url.t

  ⟨signature Www.is_active_cnx 92h⟩
  ⟨signature Www.add_active_cnx 92i⟩
  ⟨signature Www.rem_active_cnx 93a⟩
  ⟨type Www.aborter 202b⟩
```

F.2.14 www/www.ml

```
⟨www/www.ml 291a⟩≡

  ⟨type Www.request 18f⟩

  ⟨exception Www.Invalid_request 252f⟩

  ⟨constant Www.sp 19d⟩

  ⟨function Www.make 19c⟩

  ⟨module Www.UrlSet 92f⟩

  ⟨constant Www.active_connexions 92g⟩
  ⟨functions Www.xxx_active_cnx 93b⟩

  ⟨type Www.aborter 202b⟩
```

F.2.15 www/document.mli

```
⟨www/document.mli 291b⟩≡
  ⟨type Document.document_id 20a⟩

  ⟨signature Document.no_stamp 20b⟩
  ⟨signature Document.new_stamp 20e⟩

  ⟨signature type Document.logger 245a⟩
  ⟨signature Document.tty_logger 245b⟩

  ⟨type Document.handle 22b⟩

  ⟨type Document.document_continuation 22a⟩

  ⟨type Document.document_data 19f⟩

  ⟨type Document.document 19e⟩

  module DocumentIDSet : Set.S with type elt = id

  ⟨signature Document.dclose 95c⟩

  val add_log_backend: (handle -> string -> (unit -> unit) -> unit) ref

  ⟨signature Document.add_log 95d⟩
  ⟨signature Document.put_log 95e⟩
  ⟨signature Document.progress_log 95f⟩
  ⟨signature Document.end_log 95g⟩
  ⟨signature Document.destroy_log 95h⟩

  ⟨signature Document.document_id 20g⟩
```

F.2.16 www/document.ml

```
⟨www/document.ml 291c⟩≡
```

```
  open Feed
  open Www
  open Hyper

  ⟨type Document.logger ??⟩

  ⟨type Document.document_id 20a⟩

  ⟨module Document.DocumentIDSet 20i⟩

  ⟨type Document.handle 22b⟩

  ⟨type Document.document_continuation 22a⟩

  ⟨type Document.document_data 19f⟩

  ⟨type Document.document 19e⟩

  ⟨constant Document.stamp_counter 20d⟩
  ⟨constant Document.no_stamp 20c⟩

  ⟨function Document.new_stamp 20f⟩

  ⟨function Document.document_id 20h⟩


  ⟨function Document.dclose 95i⟩


  ⟨constant Document.tty_logger 245c⟩

  let add_log_backend = ref (fun _ _ _ -> failwith "no add_log defined")

  ⟨function Document.add_log 96a⟩
  ⟨functions Document.xxx_log 96c⟩
  ⟨function Document.end_log 96b⟩
```

F.3 html/

F.3.1 html/dtd.mli

```
⟨html/dtd.mli 292⟩≡
  module Elements : Set.S with type elt = string

  ⟨type Dtd.t 68b⟩

  ⟨signature Dtd.dtd20 68c⟩
  ⟨signature Dtd.dtd32 68d⟩
  ⟨signature Dtd.dtd32f 238f⟩

  ⟨signature Dtd.get 74c⟩
  ⟨signature Dtd.add 74b⟩
  ⟨signature Dtd.name 68e⟩

  ⟨signature Dtd.names 74d⟩

  ⟨signature Dtd.current 74e⟩
  ⟨signature Dtd.dump 249c⟩
```

F.3.2 html/dtd.ml

```
⟨html/dtd.ml 293a⟩≡
  open Printf

  ⟨module Dtd.elements 68a⟩

  ⟨type Dtd.t 68b⟩

  ⟨function Dtd.name 68f⟩

  ⟨function Dtd.sol 68g⟩
  ⟨function Dtd.sos 68h⟩

  ⟨constant Dtd.dtd20 68i⟩

  ⟨function Dtd.dump 249d⟩



  ⟨constant Dtd.dtd32 72⟩

  ⟨constant Dtd.current 74f⟩

  ⟨constant Dtd.table 74a⟩

  ⟨function Dtd.add 74g⟩
  ⟨constant Dtd.get 74h⟩

  ⟨function Dtd.names 74i⟩

  ⟨toplevel Dtd._1 75a⟩


  ⟨constant Dtd.dtd32f 238g⟩

  ⟨toplevel Dtd._2 239a⟩
```

F.3.3 html/html.mli

```
⟨signature Html.beautify2 293b⟩≡                                      (293d)
  val beautify2 : string -> string
    (* [beautify2 s] removes leading/trailing space and sequences of SP *)

⟨signature Html.issp 293c⟩≡                                           (293d)
  val issp : string -> bool
    (* [issp s] is true if s is formed only of SP *)

⟨html/html.mli 293d⟩≡
  (* HTML tokens *)

  ⟨type Html.attribute_name 26a⟩
  ⟨type Html.attribute_value 26b⟩
  ⟨type Html.attributes 25e⟩

  ⟨type Html.tag 25d⟩
```

```
⟨type Html.token 25c⟩

⟨type Html.location 59c⟩

⟨exception Html.Html_Lexing 252g⟩
⟨exception Html.Invalid_Html 252h⟩


⟨signature Html.init 67e⟩

⟨signature Html.verbose 245d⟩

⟨signature Html.warning 245g⟩

⟨signature Html.print 249e⟩

⟨signature Html.beautify 64c⟩

⟨signature Html.beautify2 293b⟩

⟨signature Html.issp 293c⟩

⟨signature Html.get_entity 67a⟩

val utf8_of_codepoint : int -> string
  (* [utf8_of_codepoint 233] returns the UTF-8 encoding of U+00E9 *)

⟨signature Html.get_attribute 133b⟩

⟨signature Html.has_attribute 134a⟩

⟨type Html.length 151b⟩

⟨signature Html.length_of_string 151f⟩
```

F.3.4 html/html.ml

```
⟨function Html.beautify2 294a⟩≡                                        (295)
  (* Remove also trailing space. Used for OPTION tags and TITLE *)
  let beautify2 s =
    let s1 = beautify true s in
     match String.length s1 with
       0 | 1 -> s1
     | n -> if s1.[n-1] = ' ' then String.sub s1 0 (n-1) else s1

⟨function Html.issp 294b⟩≡                                             (295)
  (* Is SP: when a PCData is only spaces, we skip it *)
  let issp s =
    try
      for i = 0 to String.length s - 1 do
        match s.[i] with
         ' '|'\t'|'\r'|'\n'|'\000' -> ()
        | _ -> failwith "subliminal"
      done;
      true
    with
      Failure "subliminal" -> false
```

```
⟨html/html.ml 295⟩≡
  open Printf

  (* HTML tokens *)

  ⟨type Html.attribute_name 26a⟩
  ⟨type Html.attribute_value 26b⟩
  ⟨type Html.attributes 25e⟩

  ⟨type Html.tag 25d⟩


  ⟨type Html.token 25c⟩

  ⟨type Html.location 59c⟩

  ⟨exception Html.Html_Lexing 252g⟩
  ⟨exception Html.Invalid_Html 252h⟩

  ⟨constant Html.verbose 245e⟩

  ⟨function Html.warning 245h⟩


  ⟨function Html.print 249f⟩

  ⟨function Html.beautify 64d⟩
  ⟨function Html.beautify2 294a⟩


  ⟨function Html.issp 294b⟩

  ⟨constant Html.ampersand_table 67c⟩

  (* Encode a Unicode codepoint as a UTF-8 string *)
  let utf8_of_codepoint n =
    if n < 0x80 then
      String.make 1 (Char.chr n)
    else if n < 0x800 then begin
      let b = Bytes.create 2 in
      Bytes.set b 0 (Char.chr (0xC0 lor (n lsr 6)));
      Bytes.set b 1 (Char.chr (0x80 lor (n land 0x3F)));
      Bytes.to_string b
    end else if n < 0x10000 then begin
      let b = Bytes.create 3 in
      Bytes.set b 0 (Char.chr (0xE0 lor (n lsr 12)));
      Bytes.set b 1 (Char.chr (0x80 lor ((n lsr 6) land 0x3F)));
      Bytes.set b 2 (Char.chr (0x80 lor (n land 0x3F)));
      Bytes.to_string b
    end else if n < 0x110000 then begin
      let b = Bytes.create 4 in
      Bytes.set b 0 (Char.chr (0xF0 lor (n lsr 18)));
      Bytes.set b 1 (Char.chr (0x80 lor ((n lsr 12) land 0x3F)));
      Bytes.set b 2 (Char.chr (0x80 lor ((n lsr 6) land 0x3F)));
      Bytes.set b 3 (Char.chr (0x80 lor (n land 0x3F)));
      Bytes.to_string b
    end else " "

  ⟨constant Html.latin1_normal 67g⟩
  ⟨function Html.init 67f⟩

  ⟨constant Html.get_entity 67b⟩


  ⟨constant Html.default_attributes 133a⟩

  ⟨function Html.get_attribute 133c⟩
  ⟨function Html.has_attribute 134b⟩

  ⟨type Html.length 151b⟩

  ⟨function Html.length_of_string 152a⟩
```

F.3.5 html/lexhtml.mli

```
⟨html/lexhtml.mli 296a⟩≡
  ⟨signature Lexhtml.strict 67h⟩

  ⟨signature type Lexhtml.t 59e⟩
  ⟨signature Lexhtml.new_data 59f⟩

  ⟨type Lexhtml.warnings 59d⟩

  ⟨signature Lexhtml.html 59a⟩
  ⟨signature Lexhtml.cdata 59b⟩
```

F.3.6 html/lexhtml.mll

```
⟨html/lexhtml.mll 296b⟩≡
  (* An HTML lexer *)
  {
  open Html

  ⟨type Lexhtml.tagtoken 61c⟩

  ⟨type Lexhtml.t 59g⟩

  ⟨function Lexhtml.new_data 59h⟩

  ⟨global Lexhtml.strict 67i⟩

  ⟨type Lexhtml.warnings 59d⟩

  ⟨helper functions Lexhtml.xxx 59i⟩

  let numeric_entity_to_utf8 code =
    try
      let n = int_of_string code in
      if n > 0x10FFFF then " " else Html.utf8_of_codepoint n
    with Failure _ -> " "
  }

  ⟨function Lexhtml.html 60a⟩


  ⟨function Lexhtml.lenient_end_comment 60d⟩
  ⟨function Lexhtml.comment 61a⟩

  ⟨function Lexhtml.next_comment 61b⟩


  ⟨function Lexhtml.text 65b⟩

  ⟨function Lexhtml.ampersand 66c⟩

  ⟨function Lexhtml.opentag 61e⟩
  ⟨function Lexhtml.closetag 62b⟩

  ⟨function Lexhtml.attrib 62d⟩
  ⟨function Lexhtml.tagattrib 63a⟩
  ⟨function Lexhtml.attribvalue 63b⟩


  ⟨function Lexhtml.inquote 64a⟩

  ⟨function Lexhtml.insingle 64b⟩

  ⟨function Lexhtml.skip_to_close 62c⟩

  ⟨function Lexhtml.cdata 66a⟩
```

F.3.7 html/html_eval.mli

```
⟨html/html_eval.mli 297a⟩≡
  ⟨signature Html_eval.debug 245i⟩

  ⟨type Html_eval.minimization 75d⟩

  ⟨signature Html_eval.add_html_filter 81b⟩
  ⟨signature Html_eval.sgml_lexer 75e⟩

  ⟨signature Html_eval.automat 75b⟩
```

F.3.8 html/html_eval.ml

```
⟨html/html_eval.ml 297b⟩≡
  open Printf
  open Html
  open Dtd

  ⟨type Html_eval.minimization 75d⟩

  ⟨constant Html_eval.debug 245j⟩

  ⟨exception Html_eval.CantMinimize 81a⟩

  ⟨constant Html_eval.initial 78b⟩


  ⟨function Html_eval.dump_stack 78a⟩

  ⟨function Html_eval.ominimize 78d⟩
  ⟨function Html_eval.cminimize 80⟩
  ⟨function Html_eval.is_cdata 78c⟩

  ⟨function Html_eval.sgml_lexer 76⟩

  ⟨constant Html_eval.filters 81d⟩
  ⟨function Html_eval.add_html_filter 81c⟩

  ⟨function Html_eval.sgml_lexer (html/html_eval.ml) 81e⟩

  ⟨function Html_eval.automat 75c⟩
```

F.3.9 html/htparse.ml

```
⟨html/htparse.ml 298a⟩≡
  (* Testing the HTML Lexer/evaluator *)
  open Html
  open Printf

  ⟨toplevel Htparse._1 82e⟩

  ⟨type Htparse.mode 81f⟩

  ⟨constant Htparse.verbose 82f⟩
  ⟨constant Htparse.mode 82a⟩

  ⟨function Htparse.error 82g⟩


  ⟨function Htparse.line_reporting 83a⟩

  ⟨function Htparse.html_lex 82d⟩

  ⟨function Htparse.html_nest 83b⟩

  ⟨function Htparse.html_indent 83c⟩

  ⟨function Htparse.main 82b⟩

  ⟨toplevel Htparse._2 82c⟩
```

F.4 http/

F.4.1 http/base64.mli

```
⟨http/base64.mli 298b⟩≡
  ⟨copyright header v6 14a⟩

  ⟨signature Base64.encode 223c⟩
  ⟨signature Base64.decode 222d⟩
```

F.4.2 http/base64.ml

```
⟨http/base64.ml 298c⟩≡
  ⟨copyright header v6 14a⟩

  ⟨constant Base64.index64 223a⟩
  ⟨toplevel Base64._1 223b⟩

  ⟨function Base64.decode 222e⟩

  ⟨constant Base64.char64 224a⟩
  ⟨toplevel Base64._2 224b⟩

  ⟨function Base64.encode 223d⟩
```

F.4.3 http/http_date.mli

```
⟨http/http_date.mli 299a⟩≡
  ⟨copyright header v6 14a⟩

  (* HTTP Date format *)

  ⟨type Http_date.http_time 108d⟩


  ⟨signature Http_date.expired 108e⟩

  ⟨signature Http_date.compare 108f⟩

  ⟨signature Http_date.string_of_ht 108g⟩

  ⟨signature Http_date.tm_of_ht 108h⟩
  ⟨signature Http_date.stamp_of_ht 108i⟩

  ⟨signature Http_date.ht_of_stamp 109a⟩
```

F.4.4 http/http_date.ml

```
⟨http/http_date.ml 299b⟩≡
  ⟨copyright header v6 14a⟩

  open Printf
  open Unix

  open Date

  ⟨type Http_date.http_time 108d⟩

  ⟨function Http_date.expired 109b⟩

  ⟨function Http_date.compare 109c⟩

  ⟨function Http_date.string_of_ht 109d⟩

  ⟨function Http_date.tm_of_ht 109e⟩

  ⟨function Http_date.stamp_of_ht 109f⟩


  ⟨function Http_date.ht_of_stamp 109g⟩
```

F.4.5 http/messages.mli

```
⟨http/messages.mli 300a⟩≡
  ⟨copyright header v6 14a⟩

  (* HTTP Messages *)

  ⟨type Messages.request 22d⟩

  ⟨type Messages.status 23b⟩

  ⟨type Messages.header 23c⟩

  (* HTTP messages: requests and responses
   *  What a client sends to a server is called a request
   *  What a server answers is called a response
   *)

  ⟨type Messages.request_message 22c⟩

  ⟨type Messages.response_message 23a⟩
```

F.4.6 http/http_headers.mli

```
⟨signature Http_headers.location 300b⟩≡                                  (300g)
  val location : Messages.header list -> string
    (* Location *)

⟨signature Http_headers.rem_contentencoding 300c⟩≡                       (300g)
  val rem_contentencoding : Messages.header list -> Messages.header list

⟨signature Http_headers.status_msg 300d⟩≡                                (300g)
  val status_msg : Messages.header list -> string

⟨signature Http_headers.status_message 300e⟩≡                            (300g)
  val status_message : int -> string
    (* [status_message n] returns Reason-Phrase for code [n] *)

⟨signature Http_headers.hints 300f⟩≡                                     (300g)
  val hints : string -> Messages.header list

⟨http/http_headers.mli 300g⟩≡

  ⟨exception Http_headers.Invalid_HTTP_header 252i⟩

  ⟨signature Http_headers.parse_status 84c⟩

  ⟨signature Http_headers.parse_request 84a⟩

  ⟨signature Http_headers.get_header 86a⟩

  ⟨signature Http_headers.get_multi_header 86c⟩

  ⟨signature Http_headers.merge_headers 107c⟩

  ⟨signature Http_headers.remove_headers 108b⟩

  ⟨signature Http_headers.header_type 107e⟩
```

```
⟨signature Http_headers.contenttype 87a⟩
⟨signature Http_headers.contentlength 88d⟩
⟨signature Http_headers.contentencoding 88f⟩
⟨signature Http_headers.location 300b⟩
⟨signature Http_headers.challenge 230b⟩
⟨signature Http_headers.proxy_challenge 230c⟩
⟨signature Http_headers.expires 230d⟩

⟨signature Http_headers.rem_contentencoding 300c⟩

⟨signature Http_headers.status_msg 300d⟩

⟨signature Http_headers.http_status 85b⟩
⟨signature Http_headers.status_message 300e⟩

(*
 * Details for specific headers
 *)

⟨type Http_headers.authScheme 230e⟩

⟨type Http_headers.authChallenge 230f⟩

⟨type Http_headers.media_parameter 23e⟩
⟨type Http_headers.media_type 23d⟩

⟨type Http_headers.hint 116b⟩

⟨signature Http_headers.hints 300f⟩

⟨signature Http_headers.read_suffix_file 117e⟩
```

F.4.7 http/http_headers.ml

⟨*function* Http_headers.status_msg 301a⟩≡ (302)

```
let rec status_msg = function
    [] -> raise Not_found
  | s::l -> if String.length s >= 5 (* "HTTP/" *)
           && (String.sub s 0 5) = "HTTP/"
          then (parse_status s).status_message
          else status_msg l
```

⟨*constant* Http_headers.is_contentencoding 301b⟩≡ (302)

```
let is_contentencoding =
  let l = String.length "Content-Encoding" in
  (fun s ->
       String.length s >= l + 2
    && String.lowercase_ascii (String.sub s 0 (l+2)) = "content-encoding: ")
```

⟨*function* Http_headers.rem_contentencoding 301c⟩≡ (302)

```
let rec rem_contentencoding = function
    [] -> []
  | h::l when is_contentencoding h -> l
  | x::l -> x :: rem_contentencoding l
```

```
⟨http/http_headers.ml 302⟩≡
  open Printf
  open Str
  open Mstring
  open Messages

  ⟨exception Http_headers.Invalid_HTTP_header 252i⟩

  ⟨function Http_headers.parse_status 85a⟩

  ⟨function Http_headers.parse_request 84b⟩


  ⟨function Http_headers.get_header 86b⟩

  ⟨function Http_headers.get_multi_header 86d⟩

  ⟨function Http_headers.header_type 108a⟩

  ⟨function Http_headers.merge_headers 107d⟩

  ⟨function Http_headers.remove_headers 108c⟩

  ⟨function Http_headers.status_msg 301a⟩


  ⟨functions Http_headers.xxx get_header applications 87b⟩

  let location =
    get_header "location"
  let challenge =
    get_header "www-authenticate"
  let proxy_challenge =
    get_header "proxy-authenticate"

  let expires hs =
    try Some (Lexdate.ht_of_string (get_header "expires" hs))
    with
       Not_found -> None
     | _ -> Log.f ("warning: Can't parse Expires header ");
        None

  ⟨constant Http_headers.is_contentencoding 301b⟩

  ⟨function Http_headers.rem_contentencoding 301c⟩

  (*
   * Details for specific headers
   *)

  ⟨type Http_headers.authScheme 230e⟩

  ⟨type Http_headers.authChallenge 230f⟩

  (* Media types *)
  ⟨type Http_headers.media_parameter 23e⟩
  ⟨type Http_headers.media_type 23d⟩

  ⟨type Http_headers.hint 116b⟩
  ⟨constant Http_headers.suffixes 116d⟩

  ⟨function Http_headers.read_suffix_file 117f⟩

  ⟨constant Http_headers.default_hints 116c⟩

  ⟨toplevel Http_headers._1 117a⟩

  ⟨function Http_headers.hints 118a⟩


  ⟨constant Http_headers.status_messages 85e⟩
  ⟨toplevel Http_headers._2 85f⟩

  ⟨function Http_headers.status_message 85d⟩

  ⟨function Http_headers.http_status 85c⟩
```

F.4.8 http/auth.mli

```
⟨http/auth.mli 303a⟩≡

  ⟨type Auth.authSpace 230g⟩

  ⟨signature Auth.lifetime 230h⟩
  ⟨signature Auth.auth_file 230i⟩

  val edit_backend: (unit -> unit) ref

  (* pad: only for edit_backend *)
  type authEntry = {
     auth_cookie : string;
     mutable auth_lastused : float
     }
  val authorizations: (authSpace, authEntry) Hashtbl.t

  ⟨signature Auth.edit 230j⟩
  ⟨signature Auth.load 231a⟩
  ⟨signature Auth.save 231b⟩

  ⟨signature Auth.add 231c⟩
  ⟨signature Auth.get 231d⟩

  ⟨signature Auth.init 233e⟩

  val open_passwd_ref: (string -> string * string) ref

  ⟨signature Auth.check 231e⟩
```

F.4.9 http/auth.ml

```
⟨http/auth.ml 303b⟩≡
  (* HTTP Basic Authentication *)

  open I18n
  open Unix
  open Http_headers
  open Www
```

```
⟨type Auth.authSpace 230g⟩

⟨type Auth.authEntry 231f⟩

⟨constant Auth.authorizations 231g⟩


⟨function Auth.get 231h⟩

⟨constant Auth.lifetime 231i⟩


⟨function Auth.lookup 231j⟩

let open_passwd_ref = ref (fun _ -> failwith "no Auth.open_passswd defined")
⟨function Auth.ask_cookie 231k⟩

⟨function Auth.replace 232a⟩


⟨function Auth.add 232b⟩

⟨function Auth.check 232c⟩

let edit_backend = ref (fun _ -> failwith "no Auth.edit defined")

(* Authorisation control *)
⟨function Auth.edit 233a⟩

⟨constant Auth.auth_file 233b⟩

⟨function Auth.save 233c⟩

⟨function Auth.load 233d⟩

⟨function Auth.init 233f⟩
```

F.4.10 http/lexheaders.mli

```
⟨http/lexheaders.mli 304a⟩≡

  ⟨signature Lexheaders.media_type 87c⟩
  ⟨signature Lexheaders.challenge 230a⟩
```

F.4.11 http/lexheaders.mll

```
⟨http/lexheaders.mll 304b⟩≡
  {
  open Http_headers


  (*
      CHAR = ['\000'-'\126']
      CTL  = ['\000'-'\031' '\127']
      CHAR except CTL = ['\032'-'\126']
      tspecials = ['(' ')' '<' '>' '@' ',' ';' ':' '\\' '"' '/' '[' ']' '?' '='
                   ' ' '\t']
  *)

  }

  rule challenge = parse
   | [^ ' ' '\t' '\r' '\n']+
      { let scheme_name = String.lowercase_ascii (Lexing.lexeme lexbuf) in
        let scheme =
      match scheme_name with
        "basic" -> AuthBasic
      | _ -> AuthExtend scheme_name in
        let _ = lws lexbuf in
        let _ = starlws lexbuf in
        let realm = realm lexbuf in
        let params = authparam lexbuf in
          { challenge_scheme = scheme;
        challenge_realm = realm;
        challenge_params = params}
      }

   | _ { raise (Http_headers.Invalid_header "auth-scheme expected")}

  and quotedstring = parse
     '"' [^ '"' '\000'-'\031' '\127'-'\255' ]* '"'
       { let t = Lexing.lexeme lexbuf in
           String.sub t 1 (String.length t - 2)
       }

   | _ { raise (Invalid_header "quotedstring expected") }

  ⟨function Lexheaders.token 87f⟩

  ⟨function Lexheaders.value 88c⟩

  (* LWS *)
  and lws = parse
     ("\r\n")? [' ' '\t']+ { () }
    | _ { raise (Invalid_header "LWS expected")}

  ⟨function Lexheaders.starlws 87g⟩

  and realm = parse
     ['R' 'r']['E' 'e']['A' 'a']['L' 'l']['M' 'm']'='
       { quotedstring lexbuf }
   | _ { raise (Invalid_header "realm expected") }


  and authparam = parse
     ','
      { let _ = starlws lexbuf in
        let t = token lexbuf in
        let _ = lit_equal lexbuf in
        let qt = quotedstring lexbuf in
        let _ = starlws lexbuf in
          (t,qt) :: authparam lexbuf
      }
   | "" { [] }

  ⟨function Lexheaders.lit_equal 88b⟩
```

```
  and lit_slash = parse
      '/' { () }
   |  _  { raise (Invalid_header "= expected") }


  ⟨function Lexheaders.media_parameters 88a⟩

  ⟨function Lexheaders.media_type lexer 87e⟩

  {
  ⟨function Lexheaders.media_type 87d⟩
  }
```

F.4.12 http/retype.mli

```
⟨http/retype.mli 306a⟩≡
  ⟨signature Retype.f 118b⟩
```

F.4.13 http/retype.ml

```
⟨http/retype.ml 306b⟩≡
  open Document
  open Http_headers

  ⟨function Retype.f 118c⟩
```

F.4.14 http/http.mli

```
⟨http/http.mli 306c⟩≡

  ⟨exception Http.End_of_headers 105d⟩

  ⟨signature Http.read_headers 106a⟩

  ⟨exception Http.HTTP_error 252j⟩

  ⟨signature Http.req 98b⟩

  ⟨signature Http.proxy_xxx 221a⟩

  ⟨signature Http.proxy_req 215f⟩

  val always_proxy: bool ref
  val send_referer: bool ref
  val user_agent: string ref
  val timeout: int ref

  val verbose: bool ref
```

F.4.15 http/http.ml

```
⟨http/http.ml 306d⟩≡
  open Common
  open I18n

  (*****************************************************************************)
```

```
(* Prelude *)
(*****************************************************************************)
(* Retrieve an HTTP document *)

(*****************************************************************************)
(* Globals *)
(*****************************************************************************)

⟨constant Http.always_proxy 221f⟩
⟨constant Http.timeout 102a⟩

⟨global Http.proxy 221b⟩
⟨global Http.proxy_port 221c⟩

⟨constant Http.verbose 245k⟩

let () = Ssl.init ()
let ssl_ctx =
  let ctx = Ssl.create_context Ssl.TLSv1_2 Ssl.Client_context in
  Ssl.set_min_protocol_version ctx Ssl.TLSv1_2;
  ctx


(*****************************************************************************)
(* Types *)
(*****************************************************************************)

⟨exception Http.HTTP_error 252j⟩

⟨type Http.status 100a⟩

⟨class Http.cnx 100b⟩

(*****************************************************************************)
(* Raw connect *)
(*****************************************************************************)

⟨function Http.tcp_connect 99⟩

(*****************************************************************************)
(* Headers *)
(*****************************************************************************)

⟨constant Http.send_referer 229a⟩
⟨constant Http.user_agent 103e⟩

⟨function Http.std_request_headers 103d⟩

(*****************************************************************************)
(* Request helpers *)
(*****************************************************************************)

⟨function Http.full_request 102e⟩

⟨function Http.failed_request 104d⟩


(*
 *  Process an HTTP request asynchronously
 *)
```

```
⟨exception Http.End_of_headers 105d⟩

⟨function Http.read_headers 106b⟩

(*****************************************************************************)
(* Responses *)
(*****************************************************************************)

⟨function Http.process_response 104e⟩

(* old: The same for HTTP 0.9, there was no header
 * so we could call directly the continuation
 * and process_response09  wwwr (cont : Document.continuation) cnx =
 *    let dh =
 *        Document.{ document_id = document_id wwwr;
 *          document_referer = wwwr.www_link.h_context;
 *          document_status = 200;
 *          dh_headers = ["Content-Type: text/html"];
 *          document_feed = Feed.of_fd cnx#fd;
 *          document_fragment = wwwr.www_fragment;
 *          document_logger = tty_logger}
 *    in
 *    cnx#set_status Discharged;
 *    cont.document_process dh
 *)

(*****************************************************************************)
(* Requests part2 *)
(*****************************************************************************)

⟨function Http.async_request 102d⟩

(* wrappers for request/response transaction *)
⟨function Http.start_request 102c⟩

⟨function Http.proxy_request 221i⟩

⟨function Http.request 98d⟩

(*****************************************************************************)
(* Entry points *)
(*****************************************************************************)
⟨function Http.req 98c⟩
⟨function Http.prox_req 215g⟩
```

F.5 protocols/

F.5.1 protocols/cache.mli

⟨*signature* Cache.history_mode 308a⟩≡ (309l)
```
  val history_mode : bool ref
```

⟨*signature* Cache.max_documents 308b⟩≡ (309l)
```
  val max_documents : int ref
```

⟨*signature* Cache.cleann 308c⟩≡ (309l)
```
  val cleann : int ref
```

⟨*type* Cache.cache_fill 309a⟩≡ (309l)
```
type cache_fill = {
  cache_write : string -> int -> int -> unit;
  cache_close : unit -> unit
 }
```

⟨*exception* Cache.DontCache 309b⟩≡ (309l)
```
exception DontCache
```

⟨*signature* Cache.tofile 309c⟩≡ (309l)
```
val tofile : Document.handle -> Document.data * cache_fill
```

⟨*signature* Cache.dummy 309d⟩≡ (309l)
```
val dummy : Document.handle  -> Document.data * cache_fill
```

⟨*signature* Cache.discard 309e⟩≡ (309l)
```
val discard: cache_fill
```

⟨*signature* Cache.wrap 309f⟩≡ (309l)
```
val wrap: cache_fill -> Document.handle -> Document.handle
```

⟨*signature* Cache.patch 309g⟩≡ (309l)
```
val patch : Document.id -> string list -> unit
```

⟨*signature* Cache.cutlinks 309h⟩≡ (309l)
```
val cutlinks : (Document.id -> unit) list ref
```

⟨*signature* Cache.make_handle 309i⟩≡ (309l)
```
val make_handle : Www.request -> Document.t -> Document.handle
```

⟨*signature* Cache.renew_handle 309j⟩≡ (309l)
```
val renew_handle : Document.handle -> Document.handle
```

⟨*signature* Cache.make_embed_handle 309k⟩≡ (309l)
```
val make_embed_handle : Document.t -> Document.handle
```

⟨protocols/cache.mli 309l⟩≡
```
(* Document and image cache *)
```

⟨*signature* Cache.debug 246b⟩
⟨*signature* Cache.history_mode 308a⟩
⟨*signature* Cache.max_documents 308b⟩
⟨*signature* Cache.cleann 308c⟩

⟨*signature* Cache.init 31g⟩

⟨*signature* Cache.add 20j⟩
⟨*signature* Cache.find 20k⟩
⟨*signature* Cache.finished 20l⟩
⟨*signature* Cache.touch 20m⟩
⟨*signature* Cache.kill 20n⟩

⟨*signature* Cache.postmortem 244b⟩

⟨*type* Cache.cache_fill 309a⟩

⟨*exception* Cache.DontCache 309b⟩

⟨*signature* Cache.tofile 309c⟩
⟨*signature* Cache.tobuffer 235c⟩
⟨*signature* Cache.dummy 309d⟩

```
  ⟨signature Cache.discard 309e⟩
  ⟨signature Cache.wrap 309f⟩

  ⟨signature Cache.patch 309g⟩

  ⟨signature Cache.cutlinks 309h⟩

  ⟨signature Cache.make_handle 309i⟩
  ⟨signature Cache.renew_handle 309j⟩
  ⟨signature Cache.make_embed_handle 309k⟩
```

F.5.2 protocols/cache.ml

```
⟨constant Cache.history_mode 310a⟩≡                                      (314e)
  let history_mode = ref false
    (* history mode means that we keep only the documents present in some
       navigator window. This mode is meant to be used in conjunction with
       a caching proxy *)

⟨constant Cache.cutlinks 310b⟩≡                                          (314e)
  (* A list of operations to do when we remove a document from the cache. *)
  let cutlinks = ref []

⟨type Cache.cache_fill (protocols/cache.ml) 310c⟩≡                       (314e)
  type cache_fill = {
    cache_write : string -> int -> int -> unit;
    cache_close : unit -> unit
   }

⟨exception Cache.DontCache (protocols/cache.ml) 310d⟩≡                   (314e)
  exception DontCache

⟨function Cache.internal_kill 310e⟩≡                                     (314e)
  (* Kills a document: stop and destroy all its dinfo
   * The caller is responsible for possible removing the document itself
   * from the memory.
   *)
  let internal_kill did _e =
     (* Remove pointers to in-lined images and other goodies *)
     List.iter (fun f -> f did) !cutlinks

⟨function Cache.make_room 310f⟩≡                                         (314e)
  let make_room () =
    if !debug then Log.f "Trying to make room in cache";
    (* Sort.list according to lru *)
    memory := List.sort
              (fun (_,e) (_,e') -> Stdlib.compare e.cache_lastused e'.cache_lastused)
           !memory;
    (* if the more recent entry has lu max_lastused, then we have to augment
       the cache, since this means that only pending connexions are
       in the cache *)
    begin match !memory with
      [] -> ()
    | (_,e)::_l ->
       if e.cache_lastused = max_lastused then max_documents := !max_documents + 5
       else (* cleanup the oldests entries *)
         let rec rem1 n l =
          if n = 0 then l
      else match l with
        [] -> []
     | (did, e)::l ->
          internal_kill did e;
          decr current;
          rem1 (n-1) l
         in
       memory := rem1 !cleann !memory
    end;
    if !debug then begin
       Log.f (sprintf "Cache size(max): %d(%d)" !current !max_documents);
       Log.f "Cache contents:";
       postmortem()
    end
```

⟨*function* Cache.finalize 311a⟩≡ (314e)

```
  (* Remove the document source. *)
  let finalize = function
     FileData (f, true) -> Msys.rm !!f
   | _ -> () (* gc ! *)
```

⟨*function* Cache.kill_entry 311b⟩≡ (314e)

```
  (* kill: removes a document from the cache
   *   Used by Reload. It can fail to find url in memory !
   *   It can also be used to remove something from the file cache
   *)
  let kill_entry did e =
    if !debug then
      Log.f (sprintf  "Killing cache entry %s(%d)"
           (Url.string_of did.document_url)
           did.document_stamp);
    internal_kill did e; (* kill dinfo in all windows *)
    finalize e.cache_document.document_data; (* remove source *)
    memory := Mlist.except_assoc did !memory;
    decr current
```

⟨*function* Cache.kill 311c⟩≡ (314e)

```
  let kill did =
    try
      let e = List.assoc did !memory in
        kill_entry did e
    with
      Not_found -> ()
```

⟨*function* Cache.finished 311d⟩≡ (314e)

```
  (* since they have lu = max_lastused *)
  let finished did =
    if !debug then
       Log.f (sprintf "%s completed" (Url.string_of did.document_url));
    try
      let entry = List.assoc did !memory in
        entry.cache_lastused <- Unix.time();
        entry.cache_pending <- false;
        Condition.set entry.cache_condition
    with
      Not_found -> ()
```

⟨*function* Cache.touch 312a⟩≡ (314e)
```
  let touch did =
    try
      let entry = List.assoc did !memory in
        entry.cache_lastused <- max (Unix.time()) entry.cache_lastused
    with
      Not_found -> ()
```

⟨*function* Cache.patch 312b⟩≡ (314e)
```
  (* Patch the headers of an existing entry *)
  let patch did headers =
    try
      let entry = List.assoc did !memory in
      let newd = {
        document_address = entry.cache_document.document_address;
        document_data = entry.cache_document.document_data;
        document_headers =
          merge_headers entry.cache_document.document_headers headers
        } in
       entry.cache_document <- newd;
      entry.cache_lastused <- max (Unix.time()) entry.cache_lastused
    with
      Not_found -> () (* is this an error ? *)
```

⟨*function* Cache.tofile 312c⟩≡ (314e)
```
  (* Cache savers *)
  let tofile _dh =
    let f = Msys.mktemp "mmmcache" in
    let oc = open_out_bin f in
      FileData (Fpath.v f,true),
        {cache_write = (fun s n1 n2 -> output oc (Bytes.of_string s) n1 n2);
         cache_close = (fun () -> close_out oc)}
```

⟨*constant* Cache.discard 312d⟩≡ (314e)
```
  let discard =
      {cache_write = (fun _buf _offs _len -> ());
       cache_close = (fun () -> ())}
```

⟨*function* Cache.dummy 312e⟩≡ (314e)
```
  (* Pseudo-caching for documents that can be obtained from the local
     file system. Relies on trailing slash for directories !
   *)

  let dummy dh =
    let url = dh.document_id.document_url in
     match url.protocol with
     | FILE ->
         begin match url.path with
         | None -> tobuffer dh
         | Some "" -> tobuffer dh
         | Some p ->
           if p.[String.length p - 1] = '/'
           then tobuffer dh
           else FileData (Fpath.v ("/"^p), false), discard
         end
     | _ -> raise DontCache
```

⟨*function* Cache.replace 313a⟩≡ (314e)
```
  let _replace = function
     MemoryData b ->
      Ebuffer.reset b;
      {cache_write = Ebuffer.output b; cache_close = (fun () -> ())}
   | FileData (f, _) ->
    let oc = open_out_bin !!f in
      {cache_write = (fun s n1 n2 -> output oc (Bytes.of_string s) n1 n2);
       cache_close = (fun () -> close_out oc)}
```

⟨*function* Cache.wrap 313b⟩≡ (314e)
```
  (* Wrap a feed with cache saving *)
  let wrap c dh =
    let wfeed = {
      feed_read =
        (fun buf offs len ->
          let r = dh.document_feed.feed_read buf offs len in
       if r <> 0 then c.cache_write (Bytes.to_string buf) offs r;
       r);
      feed_schedule = dh.document_feed.feed_schedule;
      feed_unschedule = dh.document_feed.feed_unschedule;
      feed_close =
        (fun () ->
          dh.document_feed.feed_close();
       c.cache_close();
       finished dh.document_id);
      feed_internal = dh.document_feed.feed_internal
      }
    in
    { dh with document_feed = wfeed }
```

⟨*function* Cache.fd_of_doc 313c⟩≡ (314e)
```
  (* This is stupid: to display a source that we have in the cache, we must
   * save it to disk in order to get a file descriptor...
   *)

  let fd_of_doc doc =
    match doc.document_data with
      MemoryData buf ->
        let f = Msys.mktemp "mmmbuf" in
        let oc = open_out f in
        output_string oc (Ebuffer.get buf);
        close_out oc;
        let fd = openfile f [O_RDONLY] 0 in
        Msys.rm f;
        fd
    | FileData (f,_) -> openfile !!f [O_RDONLY] 0
```

⟨*function* Cache.make_handle 313d⟩≡ (314e)
```
  let make_handle wwwr doc =
    let fd = fd_of_doc doc in
    { document_id = { document_url = wwwr.www_url; document_stamp = no_stamp};
      document_referer = wwwr.www_link.h_context;
      document_status = 200;
      dh_headers = doc.document_headers;
      document_feed = Feed.make_feed fd (Low.count_read (Unix.read fd));
      document_fragment = wwwr.www_fragment;
      document_logger = tty_logger}
```

⟨*function* Cache.renew_handle 314a⟩≡ (314e)
```
  (* The same, if we kept the old dh *)
  let renew_handle dh =
    let did = dh.document_id in
    let doc = find did in
    let fd = fd_of_doc doc in
    { document_id = dh.document_id;
      document_referer = dh.document_referer;
      document_status = dh.document_status;
      dh_headers = doc.document_headers;
      document_feed = Feed.make_feed fd (Low.count_read (Unix.read fd));
      document_fragment = dh.document_fragment;
      document_logger = dh.document_logger}
```

⟨*function* Cache.make_embed_handle 314b⟩≡ (314e)
```
  (* Same for embedded objects (but we don't have wwwr handy) *)
  let make_embed_handle doc =
    let fd =
      match doc.document_data with
        MemoryData buf ->
      let f = Msys.mktemp "mmmbuf" in
        let oc = open_out f in
          output_string oc (Ebuffer.get buf);
          close_out oc;
      let fd = openfile f [O_RDONLY] 0 in
        Msys.rm f;
        fd
      | FileData (f,_) -> openfile !!f [O_RDONLY] 0
    in
      {document_id =
      { document_url = doc.document_address; document_stamp = no_stamp};
       document_referer = None;
       document_status = 200;
       dh_headers = doc.document_headers;
       document_feed = Feed.make_feed fd (Low.count_read (Unix.read fd));
       document_fragment = None;
       document_logger = tty_logger}
```

⟨*function* Cache.cleanup 314c⟩≡ (314e)
```
  let cleanup () =
    List.iter
      (fun (_did, entry) ->
        match entry.cache_document.document_data with
         FileData (f, true) -> Msys.rm !!f
        | _ -> ())
      !memory
```

⟨*toplevel* Cache._1 314d⟩≡ (314e)
```
  let _ = at_exit cleanup
```

⟨protocols/cache.ml 314e⟩≡
```
  (* Document caching (in memory !) *)

  open Fpath_.Operators

  open Printf
  open Unix
  open Url
  open Www
  open Hyper
```

```
open Document
open Feed
open Http_headers

⟨constant Cache.debug 246c⟩
⟨constant Cache.history_mode 310a⟩

⟨constant Cache.max_lastused 21c⟩

(* The max values refer documents kept in memory *)
let max_documents = ref 30
and cleann = ref 5
and current = ref 0

⟨constant Cache.cutlinks 310b⟩

⟨type Cache.cache_fill (protocols/cache.ml) 310c⟩

⟨type Cache.entry 21b⟩

⟨exception Cache.DontCache (protocols/cache.ml) 310d⟩

⟨constant Cache.memory 21a⟩

⟨function Cache.postmortem 244c⟩


⟨function Cache.find 21d⟩

⟨function Cache.internal_kill 310e⟩

⟨function Cache.make_room 310f⟩


⟨function Cache.finalize 311a⟩

⟨function Cache.kill_entry 311b⟩

⟨function Cache.kill 311c⟩

⟨function Cache.add 235e⟩



(* Pending documents should never be removed from the cache *)
⟨function Cache.finished 311d⟩

⟨function Cache.touch 312a⟩

⟨function Cache.patch 312b⟩

⟨function Cache.init 32a⟩


⟨function Cache.tofile 312c⟩

⟨function Cache.tobuffer 235d⟩

⟨constant Cache.discard 312d⟩
```

```
  ⟨function Cache.dummy 312e⟩

  ⟨function Cache.replace 313a⟩

  ⟨function Cache.wrap 313b⟩

  (* Obtain a dh from a cache entry *)
  ⟨function Cache.fd_of_doc 313c⟩

  ⟨function Cache.make_handle 313d⟩

  ⟨function Cache.renew_handle 314a⟩


  ⟨function Cache.make_embed_handle 314b⟩


  ⟨function Cache.cleanup 314c⟩

  ⟨toplevel Cache._1 314d⟩
```

F.5.3 protocols/file.mli

```
⟨protocols/file.mli 316a⟩≡
  ⟨signature File.request 210c⟩
  ⟨exception File.File_error 210d⟩

  (* pad: for tk_file.ml *)
  val binary_path: string list ref

  ⟨signature File.pref_init 316b⟩
  ⟨signature File.pref_set 316c⟩
```

F.5.4 protocols/file.ml

```
⟨signature File.pref_init 316b⟩≡                                          (316a)

⟨signature File.pref_set 316c⟩≡                                           (316a)

⟨function File.pref_init 316d⟩≡                                           (316g)

⟨function File.pref_set 316e⟩≡                                            (316g)

⟨constant File.r 316f⟩≡                                                   (316g)
  let _r = Str.regexp ":"

⟨protocols/file.ml 316g⟩≡
  (* The file: protocol *)
  open I18n
  open Printf
  open Unix
  open Filename
  open Hyper
  open Www
  open Url
  open Messages
  open Http
  open Document
  open Feed

  ⟨exception File.File_error 210d⟩

  ⟨function File.isdir 213a⟩

  ⟨function File.d2html 212b⟩

  ⟨function File.dir 212a⟩


  ⟨function File.document_id 210e⟩

  ⟨function File.fake_cgi 239e⟩

  ⟨constant File.binary_path 239c⟩
  ⟨constant File.r 316f⟩
  ⟨function File.pref_init 316d⟩
  ⟨function File.pref_set 316e⟩

  ⟨function File.is_cgi 239d⟩
  ⟨function File.request 210f⟩
```

F.5.5 protocols/mailto.ml

```
⟨protocols/mailto.ml 317a⟩≡
  (* mailto: *)
  open I18n

  open Unix
  open Www
  open Hyper
  open Url

  ⟨constant Mailto.mailer 214a⟩

  ⟨type Mailto.msg 213d⟩

  ⟨function Mailto.error 214f⟩

  ⟨function Mailto.sendmail 214e⟩

  ⟨global Mailto.internal_backend 214d⟩

  ⟨function Mailto.internal 214c⟩

  ⟨function Mailto.get 214b⟩

  ⟨function Mailto.f 213e⟩
```

F.5.6 protocols/protos.mli

```
⟨protocols/protos.mli 317b⟩≡

  ⟨signature Protos.get 21e⟩
```

F.5.7 protocols/protos.ml

```
⟨protocols/protos.ml 318a⟩≡
  open Common

  ⟨constant Protos.protos 21f⟩

  ⟨toplevel Protos._1 215a⟩
  ⟨toplevel Protos._2 98a⟩
  ⟨toplevel Protos._3 215b⟩
  ⟨toplevel Protos._4 215c⟩
  ⟨toplevel Protos._5 215d⟩
  ⟨toplevel Protos._6 215e⟩
  ⟨toplevel Protos._7 209j⟩

  ⟨constant Protos.get 21g⟩
```

F.6 retrieve/

F.6.1 retrieve/progress.mli

```
⟨retrieve/progress.mli 318b⟩≡
  ⟨signature Progress.no_meter 97c⟩
  ⟨signature Progress.meter 97e⟩
```

F.6.2 retrieve/progress.ml

```
⟨retrieve/progress.ml 318c⟩≡
  ⟨constant Progress.no_meter 97d⟩

  ⟨function Progress.meter 97f⟩
```

F.6.3 retrieve/retrieve.mli

```
⟨retrieve/retrieve.mli 318d⟩≡
  (* Document retrieval *)

  ⟨type Retrieve.retrievalStatus 89b⟩

  ⟨signature Retrieve.f 89a⟩

  ⟨type Retrieve.behaviour 90a⟩

  ⟨signature Retrieve.add_http_processor 91c⟩
```

F.6.4 retrieve/retrieve.ml

```
⟨retrieve/retrieve.ml 318e⟩≡
  open Common
  open I18n

  (*****************************************************************************)
  (* Prelude *)
  (*****************************************************************************)
  (* Document retrieval *)
```

```
(*****************************************************************************)
(* Types *)
(*****************************************************************************)
⟨type Retrieve.retrievalStatus 89b⟩

⟨type Retrieve.behaviour 90a⟩

(*****************************************************************************)
(* Globals *)
(*****************************************************************************)
⟨constant Retrieve.http_process 91b⟩

⟨constant Retrieve.add_http_processor 92a⟩

(*****************************************************************************)
(* Cache *)
(*****************************************************************************)
⟨function Retrieve.wrap_cache 91a⟩

(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)
⟨function Retrieve.http_check 90b⟩

(*****************************************************************************)
(* Entry point *)
(*****************************************************************************)
⟨function Retrieve.f 89c⟩

(*****************************************************************************)
(* Response code -> behavior *)
(*****************************************************************************)

(* In all the following, we avoid popping up dialog boxes, and use
 * wwwr logging instead. Otherwise we might get too verbose for
 * in-lined images...
 *)

⟨function Retrieve.code200 92c⟩

⟨function Retrieve.code204 92d⟩

⟨function Retrieve.forward 222c⟩

⟨function Retrieve.forward_permanent 222b⟩

(* 304 : Response to a conditional GET, the document is not modified
let update wr dh =
   Cache.patch dh.document_id dh.dh_headers;
   Stop (s_ "Document %s has not changed.\n" (Url.string_of wr.www_url))
Because of recursive update, this has moved elsewhere.
*)

⟨function Retrieve.code400 92e⟩

⟨function Retrieve.ask_auth 234b⟩

⟨function Retrieve.unauthorized 234c⟩
```

```
  ⟨function Retrieve.ask_proxy_auth 235a⟩

  ⟨function Retrieve.proxy_unauthorized 235b⟩

  ⟨toplevel Retrieve._1 92b⟩
```

F.6.5 retrieve/img.mli

```
⟨retrieve/img.mli 320a⟩≡
  ⟨signature Img.gif_anim_load 218c⟩

  module ImageData : sig
    type t = Tkanim.imageType

    val gamma : float ref
    val jpeg_converter : string ref
    val verbose : bool ref

    val load : Document.handle -> Document.id list -> Fpath.t -> Tkanim.imageType
    val cache_access : Url.t -> Document.id -> Tkanim.imageType
    val error :
        Url.t -> (Document.id * ((Url.t -> Tkanim.imageType -> unit) * Scheduler.progress_func)) list -> unit
    val error_msg : Www.request * string -> unit
    val remove_reference : Document.id -> unit
    val dump: unit -> unit
  end

  module ImageScheduler : Scheduler.S with
      type shared_data = ImageData.t

  ⟨signature Img.get 96d⟩
  ⟨signature Img.update 96e⟩
```

F.6.6 retrieve/img.ml

```
⟨toplevel Img._1 320b⟩≡                                                    (320c)
  (* Advertise ourselfs to the internal cache *)
  let _ =
   Cache.cutlinks := ImageData.remove_reference :: !Cache.cutlinks

⟨retrieve/img.ml 320c⟩≡
  (* Image cache and scheduled image downloading *)

  open Fpath_.Operators
  open Tk

  ⟨constant Img.gif_anim_load 218d⟩

  module ImageData =
    struct

      type t = Tkanim.imageType

      let gamma = ref 1.0
      let jpeg_converter = ref "convert"
      let verbose = ref false

     (*
* The image cache
*)

let set_of_list l =
  List.fold_right Document.DocumentIDSet.add l Document.DocumentIDSet.empty

(* url -> (option for tk configure, set of referers, headers) *)
let img_cache =
   (Hashtbl.create 53 : (Url.t,
             Tkanim.imageType * Document.DocumentIDSet.t ref
               * string list) Hashtbl.t)

(* Debugging *)
let dump () =
  Hashtbl.iter (fun url (_,r, _) ->
   Logs.debug (fun m -> m "IMG %s" (Url.string_of url));
   Document.DocumentIDSet.iter
     (fun did -> Logs.debug (fun m -> m "\tref: %s"
                     (Url.string_of did.document_url)))
     !r)
img_cache

let add url imgdesc referers headers =
  Hashtbl.add img_cache url (imgdesc, ref (set_of_list referers), headers)

(* Raises Not_found *)
let cache_access url from =
  let img, refs, _ = Hashtbl.find img_cache url in
refs := Document.DocumentIDSet.add from !refs;
img

let direct_cache_access  = Hashtbl.find img_cache

(* Delete an image from the cache *)
let delete_image img =
  if !verbose
  then Logs.info (fun m -> m "Removing img %s" (Url.string_of img));
  match Hashtbl.find img_cache img with
Still x, _, _ ->
  begin match x with
    Bitmap _ -> ()
  | ImageBitmap n ->
      Imagebitmap.delete n; Hashtbl.remove img_cache img
  | ImagePhoto n ->
      Imagephoto.delete n; Hashtbl.remove img_cache img
  | _ -> assert false
  end
  | Animated anm, _, _ -> Tkanim.delete anm; Hashtbl.remove img_cache img

(* Remove reference to an image, clean *)
let remove_reference (referer : Document.id) =
  if !verbose
  then Logs.info (fun m -> m "Removing img references from %s(%d)"
        (Url.string_of referer.document_url)
         referer.document_stamp);
  let delete_them = ref [] in
  Hashtbl.iter
(fun img (_o, refs, _) ->
    refs := Document.DocumentIDSet.remove referer !refs;
    if Document.DocumentIDSet.is_empty !refs then
```

```
     delete_them := img :: !delete_them)
img_cache;
  List.iter delete_image !delete_them

let broken_data = Tkanim.Still (Bitmap (Predefined "error"))

(* load an image *)
(* For GIFs, we use JPF's Tkanim package first *)
let tk_load_gif file =
  try
if !gif_anim_load then Tkanim.create file
else Still (ImagePhoto (Imagephoto.create [File file; Gamma !gamma]))
  with Protocol.TkError _ -> broken_data

(* For JPEG, we attempt internal load first (works when tkimg/libtk-img is
   installed and loaded via 'package require Img' at startup).
   If that fails, fall back to an external converter (default: ImageMagick
   'convert') to produce a PPM file that Tk can always read. *)
let tk_load_jpeg file =
  try Tkanim.Still (ImagePhoto (Imagephoto.create [File file; Gamma !gamma]))
  with Protocol.TkError _ ->
let ppmfile = (Msys.mktemp "img") ^ ".ppm" in
let cmd = (!jpeg_converter ^ " " ^ file ^ " " ^ ppmfile) in
try match Sys.command cmd with
  0 ->
    let img = Tkanim.Still (ImagePhoto (Imagephoto.create
                 [File ppmfile; Gamma !gamma])) in
    Msys.rm ppmfile;
    img
| _ -> Msys.rm ppmfile; broken_data
with
  Protocol.TkError _ ->
    Msys.rm ppmfile;
    Still (Bitmap (Predefined "question"))

(* other formats *)
let tk_load_other file =
  Tkanim.Still (
    try ImageBitmap (Imagebitmap.create [File file])
with
  Protocol.TkError _ ->
    try ImagePhoto (Imagephoto.create [File file; Gamma !gamma])
    with
     Protocol.TkError _ -> Bitmap (Predefined "question"))

let load dh referers (file : Fpath.t) =
  Retype.f dh;
  match dh.document_status with
200 ->
  let url = dh.document_id.document_url in
  let img =
    try
      let ctype = Http_headers.contenttype dh.dh_headers in
      match Lexheaders.media_type ctype with
       ("image","jpeg"), _ -> Low.busy tk_load_jpeg !!file
      | ("image","gif"), _ -> Low.busy tk_load_gif !!file
      | _,_ -> Low.busy tk_load_other !!file
    with
    | Not_found -> Low.busy tk_load_other !!file
    | Http_headers.Invalid_header _ -> Msys.rm !!file; broken_data
```

```
            in
        if !verbose
        then Logs.info (fun m -> m "Loaded %s as %s" !!file (Url.string_of url));
        Msys.rm !!file;
        add url img referers dh.dh_headers;
        img
        | 304 -> (* we did an update an a document, and it induced a
            recursive update. The document didn't change *)
        begin try
          Msys.rm !!file;
          cache_access dh.document_id.document_url (List.hd referers)
        with
          Not_found -> broken_data
        end

        | _ -> (* other cases *)
        Msys.rm !!file; broken_data

        (* error during img downloading *)
      let error url job =
        Logs.err (fun m -> m "Could not load image at %s" (Url.string_of url));
        let img = Tkanim.Still (Bitmap (Predefined "error")) in
        add url img (List.map fst job) [];
        List.iter (fun (_, (cont,_)) -> cont url img) job

        (* Invalid urls in images are silently ignored *)
      let error_msg ((w : Www.request), msg) =
        Logs.err (fun m -> m "Invalid image request: %s (%s)"
             (Url.string_of w.www_url) msg);

    end


  module ImageScheduler = Scheduler.Make(ImageData)


  ⟨toplevel Img._1 320b⟩

  ⟨function Img.get 96f⟩

  ⟨function Img.update 96g⟩
```

F.6.7 retrieve/scheduler.mli

```
⟨retrieve/scheduler.mli 323⟩≡
  ⟨toplevel comment Scheduler 97a⟩

  ⟨signature Scheduler.debug 246e⟩

  ⟨type Scheduler.progress_func 97b⟩

  module type Data =
    sig
     type t
          (* Type of shared objects
           * The table of objects in managed in this module
           *)
     val load : Document.handle -> Document.id list -> Fpath.t -> t
          (* [load dh referers file]
           *   is responsible for creating the shared handle
           *)
     val cache_access : Url.t -> Document.id -> t
          (* [cache_access url referer]
           *   attempts to find a shared handle for an URL.
           *   Raises Not_found
           *)
     val error :
          Url.t ->
          (Document.id * ((Url.t -> t -> unit) * progress_func)) list -> unit
          (* [error url [(did,(cont,progress))]]
           *  if an error occurs, then each pending continuation is called
           *  (if necessary) as required (e.g. with "default" information)
           *)
     val error_msg : Www.request * string -> unit
        (* Retrieval produces Invalid_url *)
    end


  module type S =
    sig
     type shared_data

     (* ?? -> <> -> Retrieve.f -> Http.req (via protos) *)
     val add_request : < Cap.network > ->
          Www.request -> Document.id ->
          (Url.t -> shared_data -> unit) -> progress_func -> unit
          (* [add_request delayed wr referer cont progress_func]
           *   returns job handle that can subsequently by awakened
           *)

     val stop : Document.id -> unit
          (* [stop did]
           *   stops jobs for which did is the only referer
           *)

     (* Delayed queues for this scheduler *)
     type delayed

     val new_delayed : unit -> delayed

     val add_delayed :
          delayed -> Www.request -> Document.id ->
              (Url.t -> shared_data -> unit) -> progress_func -> unit

     val flush_delayed : delayed -> unit
     val flush_one : < Cap.network > -> delayed -> Url.t -> unit

     val is_empty : delayed -> bool
     val maxactive : int ref
     val maxsamehost : int ref
    end

  module Make(J : Data):(S with type shared_data = J.t)
```

F.6.8 retrieve/scheduler.ml

⟨retrieve/scheduler.ml 324⟩≡
```
  (* Scheduled downloading *)
```

```
⟨constant Scheduler.debug 246f⟩

⟨type Scheduler.progress_func 97b⟩

(* Handling of data downloaded by this scheduler *)
module type Data = sig
  type t

  val load : Document.handle -> Document.id list -> Fpath.t -> t
  (* [load dh referers file] *)

  val cache_access : Url.t -> Document.id -> t
  (* [cache_access url referer] *)

  val error :
    Url.t -> (Document.id * ((Url.t -> t -> unit) * progress_func)) list -> unit
  (* [error url conts] *)

  val error_msg : Www.request * string -> unit
  (* Retrieval produces Invalid_url *)
end

module type S = sig
  type shared_data

  val add_request :
    < Cap.network > ->
    Www.request ->
    Document.id ->
    (Url.t -> shared_data -> unit) ->
    progress_func ->
    unit
  (* [add_request wwwr ref_did cont progress_func] *)

  val stop : Document.id -> unit
  (* [stop ref_did] *)

  (* Delayed queues for this scheduler *)
  type delayed

  val new_delayed : unit -> delayed

  val add_delayed :
    delayed ->
    Www.request ->
    Document.id ->
    (Url.t -> shared_data -> unit) ->
    progress_func ->
    unit

  val flush_delayed : delayed -> unit
  val flush_one : < Cap.network > -> delayed -> Url.t -> unit
  val is_empty : delayed -> bool
  val maxactive : int ref
  val maxsamehost : int ref
end

module Make (J : Data) = struct
  type shared_data = J.t
```

```
let maxactive = ref 10
let maxsamehost = ref 2

(* A job is: a list of referers, with the continuations *)
type job = {
  mutable stop : unit -> unit;
  mutable conts :
    (Document.id * ((Url.t -> shared_data -> unit) * progress_func)) list;
  mutable bytes_loaded : int;
  mutable contentlength : int option;
}

(* The list of active requests : this is used to share the requests
   for all jobs on the same Url. *)
let active = ref 0
and actives = (Hashtbl.create 11 : (Url.t, job) Hashtbl.t)

(* We need a two-level queue system, so that
   1- we respect the image loading order for each document
   2- we can use maxactive connexions
   3- there is a max of maxsamehost connexions on the same host
*)

let samehost = (Hashtbl.create 11 : (string, int ref) Hashtbl.t)
(* count of cnx on each host (IP number is best choice), but for
   performance reason (DNS lookups), we take FQDN *)

let addhost (url : Url.t) =
  let s =
    match url.host with
    | Some s -> s
    | None -> ""
  in
  try
    let count = Hashtbl.find samehost s in
    if !count < !maxsamehost then (
      incr count;
      true)
    else false
  with
  | Not_found ->
      Hashtbl.add samehost s (ref 1);
      true (* assumes maxsamehost >= 1 *)

let remhost (url : Url.t) =
  let s =
    match url.host with
    | Some s -> s
    | None -> ""
  in
  try
    let r = Hashtbl.find samehost s in
    decr r;
    if !r <= 0 then Hashtbl.remove samehost s
  with
  | Not_found -> () (* that's an error actually *)

type queue =
  (Www.request * Document.id * (Url.t -> shared_data -> unit) * progress_func)
  Queue.t
(* queue for one batch of docs *)

let queues = (ref [] : queue list ref)
(* pending queues for documents *)

(* How we pick the next request *)

exception Busy

let skip_cache (wr : Www.request) =
  try Http_headers.get_header "pragma" wr.www_headers = "no-cache" with
  | Not_found -> false

let pick () =
  let pick_in_batch q =
    try
      let (wr : Www.request), _, _, _ = Queue.peek q in
      let url = wr.www_url in
      if addhost url then Some (Queue.take q) else None
    with
    | Queue.Empty ->
        (* this batch is empty *)
        raise Queue.Empty
  in
  let rec walk_batches remaining = function
    | [] ->
        (* we've reached the end : reset the remaining scheduled jobs *)
        queues := List.rev remaining;
        raise Busy
    | x :: l -> (
        try
          match pick_in_batch x with
          | Some r -> r
          | None ->
              (* nothing pickable yet, look further *)
              walk_batches (x :: remaining) l
        with
        | Queue.Empty ->
            (* this queue is empty ! *)
            walk_batches remaining l)
  in
  walk_batches [] !queues

(* Whenever we add something in the queue, we must call this *)
(* Whenever a job finished, we must call this *)
let rec next_request caps =
  if !active < !maxactive then
    try
      let j = pick () in
      process_request caps j;
      next_request caps (* check if more can be done *)
    with
    | Busy -> ()

(* when adding a request individually (meant to be treated ASAP), we
   use a new singleton queue *)
and add_request (caps : < Cap.network ; .. >) (wr : Www.request)
    (did : Document.id) cont (prog : progress_func) =
  let q = Queue.create () in
  Queue.add (wr, did, cont, prog) q;
  queues := q :: !queues;
  next_request caps

(* error during data downloading *)
and error (caps : < Cap.network; .. >) (url : Url.t) (job : job) =
  job.stop ();
  J.error url job.conts;
  if !debug then
    Logs.warn (fun m -> m "Retrieval of %s failed\n" (Url.string_of url));
  next_request caps

(* process_request always follows pick, thus hostcount has always been
 * incremented for the URL of this request *)
and process_request (caps : < Cap.network ; .. >) (wr, did, cont, prog) =
  try
    (* if we are in the cache of shared objects, apply continuation *)
    if skip_cache wr then raise Not_found
    else begin
      let data = J.cache_access wr.www_url did in
      remhost wr.www_url;
      (* we're done *)
      cont wr.www_url data
    end
  with
  | Not_found -> (
      (* find out if we are in the active jobs *)
      let url = wr.www_url in
      try
        let oldjob = Hashtbl.find actives url in
        (* then add a new continuation *)
        oldjob.conts <- (did, (cont, prog)) :: oldjob.conts;
        remhost wr.www_url
        (* we're done *)
      with
      | Not_found -> begin
          (* start a new job *)
          if !debug then
            Logs.debug (fun m -> m "Starting job for %s" (Url.string_of url));
          let job =
            {
              stop =
                (fun () ->
                  Hashtbl.remove actives url;
                  decr active;
                  remhost url);
              conts = [ (did, (cont, prog)) ];
              contentlength = None;
              bytes_loaded = 0;
            }
          in
          (* Add to set of active *)
          incr active;
          Hashtbl.add actives url job;

          (* We are now going to run the retrieval process *)

          (* Continuations for the retrieval *)
          (* TODO: remove those nested function, move out and pass down
           * necessary closed arguments
           *)
          let handle_data (dh : Document.handle) =
            (* add more things to do in stop *)
            let oldstop = job.stop in
            job.stop <-
              (fun () ->
                Document.dclose true dh;
                oldstop ());
            try
              (* open the temporary file in which doc is to be saved *)
              let file = Msys.mktemp "data" in
              let oc = open_out file and buffer = Bytes.create 2048 in

              (* JPF HACK -- for Image retrieval progress meter *)
              begin
                try
                  job.contentlength <-
                    Some (Http_headers.contentlength dh.dh_headers)
                with
                | Not_found -> ()
              end;

              (* actually start sucking data *)
              dh.document_feed.feed_schedule (fun () ->
                  try
                    let n = dh.document_feed.feed_read buffer 0 2048 in

                    (* JPF HACK -- for Image retrieval progress meter *)
                    job.bytes_loaded <- job.bytes_loaded + n;
                    List.iter
                      (fun (_, (_, prog)) ->
                        prog job.contentlength job.bytes_loaded)
                      job.conts;

                    if n <> 0 then output oc buffer 0 n
                    else begin
                      (* end of document *)
                      Document.dclose true dh;
                      (* see comment below *)
                      close_out oc;
                      (* proceed to load and run continuations *)
                      let referers = List.map fst job.conts in
                      begin
                        try
                          let data = J.load dh referers (Fpath.v file) in
                          List.iter
                            (fun (_referer, (cont, _)) ->
                              try
                                Printexc.print
                                  (cont dh.document_id.document_url)
                                  data
                              with
                              | _ -> flush Stdlib.stderr)
                            job.conts
                        with
                        (* load failed *)
                        | e ->
                            Logs.warn (fun m -> m "Load error %s"
                                          (Printexc.to_string e));
                            J.error url job.conts
                      end;
                      (* we must remove from active only after
                         loading because otherwise, if loading is interactive,
                         there could be a moment during which the document
                         is not marked as loaded, but not active either.
                         This would cause multiple retrievals.
                         But then dh has to be closed otherwise the
                         callback will we called indefinitely *)
                      oldstop ();
                      if !debug then
                        Logs.debug (fun m -> m
                          "Finished job for %s" (Url.string_of url));
                      (* proceed with more requests *)
                      next_request caps
                    end
                  with
                  (* errors in retrieval *)
                  | Unix.Unix_error (code, s, s') ->
                      Logs.err (fun m -> m "Unix error (%s) in scheduler %s %s"
                                            (Unix.error_message code) s s');
                      close_out oc;
                      error caps url job
                  | Sys_error s ->
                      Logs.err (fun m -> m "IO error (%s) in scheduler" s);
                      close_out oc;
                      error caps url job
                  | e ->
                      Logs.err (fun m -> m
                        "Bug in scheduler %s" (Printexc.to_string e));
                      close_out oc;
                      error caps url job)
            with
            (* error creating tmp file *)
            | Sys_error s ->
                Logs.err (fun m -> m "Can't create temporary file (%s)" s);
                error caps url job
            | e ->
                Logs.err (fun m -> m "Bug in scheduler %s" (Printexc.to_string e));
                error caps url job


          (* Data has moved. The best way to do this properly is to
             reschedule the job conts as new requests *)
          and retry_data (hlink : Hyper.link) =
            try
              job.stop ();
              let newr = Www.make hlink in
              newr.www_error <- wr.www_error;
              newr.www_logging <- wr.www_logging;
              job.conts |> List.iter (fun (did, (cont, prog)) ->
                  add_request caps newr did cont prog)
            with
            (* can't proceed with retry *)
            | _ -> error caps url job
          in


          (* Okay, go for the retrieval now *)
          try
            match
```

```
              Retrieve.f caps wr retry_data
                {
                  document_process = handle_data;
                  document_finish = (fun f -> if f then error caps url job);
                }
            with
            | Retrieve.Started _ -> ()
            | Retrieve.InUse ->
                (* somebody else has started a request bypassing the
                   scheduler, dammit. Our only hope is that he's going
                   to set the cache properly, so we can reschedule
                   ourself and try later *)
                job.stop ();
                List.iter
                  (fun (did, (cont, prog)) ->
                    add_request caps wr did cont prog)
                  job.conts
          with
          | Www.Invalid_request (w, msg) ->
              (* retrieve failed *)
              J.error_msg (w, msg);
              error caps url job
        end)

(*
 * And now, various utilities
 *)

(* remove pending requests whose referer is did *)
let stop (did : Document.id) =
  (* For all queues, for all request in the queue, if the request matches
     the predicate, it is removed from the queue. *)
  queues :=
    !queues |> List.map (fun q ->
           let newq = Queue.create () in
           q |> Queue.iter (function
                | _wr, didr, _cont, _progress when did = didr -> ()
                | r -> Queue.add r newq);
           newq);

  (* If the request is active, remove the particular continuation, and if it
     was the only continuation, kill the job
  *)
  let rem = ref [] in
  (* jobs to kill *)
  actives |> Hashtbl.iter
    (fun _url job ->
      try
        job.conts <- Mlist.except_assoc did job.conts;
        if job.conts = [] then rem := job :: !rem
      with
      | Not_found -> ()
  );
  (* each stop closes the cnx properly and remove the job from actives *)
  !rem |> List.iter (fun job -> job.stop ());
  if !rem <> [] then
    let caps = Cap.network_caps_UNSAFE () in
    next_request caps

(*
     * Delayed queues
     *)
    type delayed = queue

    let new_delayed = Queue.create

    let is_empty (q : queue) =
      try
        Queue.peek q |> ignore;
        false
      with
      | Queue.Empty -> true

    (* add a new request in the queue *)
    (* Actually, if the document is already in the cache, then process
       the continuation *)
    let add_delayed (q : delayed) (wr : Www.request) did cont progress =
      try
        if skip_cache wr then raise Not_found
        else cont wr.www_url (J.cache_access wr.www_url did)
      with
      | Not_found -> Queue.add (wr, did, cont, progress) q

    (* Put the queue in the list of queues *)
    let flush_delayed (q : delayed) : unit =
      (* Queue.iter (function (_,_,_,prog) -> prog None 0) q;(* create the gauge *) *)
      queues := !queues @ [ q ];
      let caps = Cap.network_caps_UNSAFE () in
      next_request caps

    (* Flush a particular request from a queue : we do it in place
       because we don't know if the queue has been put in the list yet
    *)
    let flush_one (caps : < Cap.network >) (l : delayed) (url : Url.t) : unit  =
      let flushedqueue = Queue.create () and restqueue = Queue.create () in
      (* split in two *)
      l |> Queue.iter (function
          | (wr : Www.request), did, cont, prog when wr.www_url = url ->
              prog None 0;
              (* create the gauge *)
              Queue.add (wr, did, cont, prog) flushedqueue
          | r -> Queue.add r restqueue
      );
      (* the flushed goes at the beginning *)
      queues := flushedqueue :: !queues;
      (* copy back the remaining in l (MUST BE THE SAME l) *)
      Queue.clear l;
      restqueue |> Queue.iter (fun r -> Queue.add r l);
      (* try to process the flushed items *)
      next_request caps
  end
```

F.7 viewers/

F.7.1 viewers/decoders.mli

⟨viewers/decoders.mli 332⟩≡
```
  (* Decoders *)
  ⟨signature Decoders.insert 217b⟩
  ⟨signature Decoders.add 217c⟩
```

F.7.2 viewers/decoders.ml

```
⟨viewers/decoders.ml 333a⟩≡
  open Unix
  open Document
  open Feed
  open Http_headers

  ⟨constant Decoders.decoders 217d⟩

  ⟨function Decoders.gzip 217f⟩

  ⟨toplevel Decoders._1 217e⟩

  ⟨constant Decoders.add 218a⟩
  ⟨function Decoders.insert 218b⟩
```

F.7.3 viewers/embed.mli

```
⟨viewers/embed.mli 333b⟩≡
  module EmbeddedScheduler : Scheduler.S with
    type shared_data = Document.t

  ⟨type Embed.viewer 164g⟩

  ⟨signature Embed.add_viewer 164h⟩

  ⟨signature Embed.rem_viewer 164i⟩

  ⟨type Embed.embobject 164a⟩

  ⟨signature Embed.add 164b⟩
  ⟨signature Embed.update 164c⟩
```

F.7.4 viewers/embed.ml

```
⟨viewers/embed.ml 333c⟩≡
  (* Embedded documents *)

  open Fpath_.Operators
  open I18n
  open Tk

  (* Assume any kind of data could be embedded
   * The normal retrieval, used by the scheduler, makes its own decision
   * about the need to cache the document (basically, it caches html and text)
   * Thus, we want to decide here if we want to cache documents retrieved
   * via Embed.
   *)
  module EmbeddedData = struct
    type t = Document.t

    let cache_access url _referer =
      let did = Document.{ document_url = url; document_stamp = no_stamp } in
      (* look in the cache *)
      Cache.find did

    (* The document is here in the file. Either it's been cached, and
       then we just get its cache value, or we add it to the cache
       dh is closed; we use only the headers
       NOTE: if we are updating over an old version, fix the cache
    *)
    let load dh referers (file : Fpath.t) =
      Retype.f dh;
      match dh.document_status with
      | 200 -> begin
          try
            let doc = Cache.find dh.document_id in
            let this_date = Http_headers.get_header "date" dh.dh_headers in
            let cache_date = Http_headers.get_header "date" doc.document_headers in
            if this_date <> cache_date then raise Not_found else doc
          with
          | Not_found ->
              let doc =
                Document.
                  {
                    document_address = dh.document_id.document_url;
                    document_data = FileData (file, true);
                    document_headers = dh.dh_headers;
                  }
              in
              Cache.add dh.document_id doc;
              Cache.finished dh.document_id;
              doc
        end
      | 304 -> begin
          (* return the previous version *)
          try
            Msys.rm !!file;
            cache_access dh.document_id.document_url (List.hd referers)
          with
          | Not_found -> failwith "load"
        end
      | _ -> failwith "load"

    let error url _jobs =
      Error.f (s_ "Can't find embedded document %s" (Url.string_of url))

    let error_msg (_, _) = ()
  end

  (* The embedded data scheduler *)
  module EmbeddedScheduler = Scheduler.Make (EmbeddedData)

  (* Embedded viewers *)

  ⟨type Embed.viewer 164g⟩

  ⟨constant Embed.embedded_viewers 164j⟩
  let add_viewer = Hashtbl.add embedded_viewers
  and rem_viewer = Hashtbl.remove embedded_viewers

  ⟨function Embed.embedded_viewer 165a⟩
```

```
  ⟨type Embed.embobject 164a⟩

  ⟨constant Embed.embedded 165b⟩

  ⟨function Embed.add_embed 165c⟩

  ⟨function Embed.when_destroyed 165d⟩
  ⟨toplevel Embed._1 165e⟩

  ⟨function Embed.add 165f⟩

  ⟨function Embed.update 167⟩
```

F.7.5 viewers/save.mli

```
⟨signature Save.transfer 335a⟩≡                                          (335d)
  val transfer : Www.request -> Document.handle -> (Unix.file_descr * bool) option -> unit

⟨signature Save.tofile 335b⟩≡                                            (335d)
  val tofile : (string -> unit) -> Document.handle -> string -> string -> unit

⟨signature Save.print_command 335c⟩≡                                     (335d)
  val print_command : string ref

⟨viewers/save.mli 335d⟩≡
  ⟨signature Save.interactive 121d⟩
  ⟨signature Save.transfer 335a⟩
  ⟨signature Save.tofile 335b⟩

  ⟨signature Save.document 45c⟩
  ⟨signature Save.print_command 335c⟩
```

F.7.6 viewers/save.ml

```
⟨function Save.f 335e⟩≡                                                  (339b)
  (* Save to file fname, and apply continuation cont to this file *)
  (* unprotected against Sys_error *)
  let f cont dh fname endmsg =
    let oc = open_out_bin fname in
    let buffer = Bytes.create 1024 in
    let red = ref 0 in
    let size =
      try Http_headers.contentlength dh.dh_headers
      with Not_found -> 40000 (* duh *)
    in
    dh.document_feed.feed_schedule
      (fun () ->
        try
      let n = dh.document_feed.feed_read buffer 0 1024 in
      if n = 0 then begin
        dclose true dh;
        close_out oc;
        Document.end_log dh endmsg;
        cont fname (* cont is responsible for deleting fname *)
      end
      else begin
            output oc buffer 0 n;
```

```
          red := !red + n;
        Document.progress_log dh (! red * 100 / size)
      end
        with
      Unix_error(_,_,_) | Sys_error _ ->
        dclose true dh;
        close_out oc;
        Document.destroy_log dh false;
        Msys.rm fname;
        Error.f (s_ "Error during retrieval of %s"
                       (Url.string_of dh.document_id.document_url))
       )
```

⟨*function* Save.tofile 336a⟩≡ (339b)

```
  (* Used for external viewers in batch mode. Deprecated *)
  let tofile cont dh fname endmsg =
    try
      f cont dh fname endmsg
    with Sys_error msg ->
      dclose true dh;
      Document.destroy_log dh false;
      Error.f (s_ "Cannot save to %s\n(%s)" fname msg)
```

⟨*function* Save.transfer 336b⟩≡ (339b)

```
  let transfer wr dh dest =
    wr.www_logging (s_ "Saving...");
    match dest with
      None -> interactive (fun _s -> wr.www_logging "") dh
    | Some (fd, flag) ->
        (* if flag we should output the headers as well *)
        if flag then begin
      dh.dh_headers |> List.rev |> List.iter (fun h ->
        Munix.write_string fd h; Munix.write_string fd "\n"
      );
      Munix.write_string fd "\n";
        end;
        let buffer = Bytes.create 1024 in
        dh.document_feed.feed_schedule
         (fun () ->
        try
          let n = dh.document_feed.feed_read buffer 0 1024 in
          if n = 0 then begin
            dclose true dh;
            close fd;
          end
       else ignore (write fd buffer 0 n)
      with
       Unix_error(_,_,_) | Sys_error _ ->
         dclose true dh;
         close fd;
         Error.f (s_"Error during retrieval of %s"
                       (Url.string_of dh.document_id.document_url))
         )
```

⟨*function* Save.save_from_string 336c⟩≡ (339b)

```
  let save_from_string url s f =
    try
      let oc = open_out_bin f in
      begin try
        output_string oc s;
        Error.ok (s_ "Document %s\nsaved in\n%s" (Url.string_of url) f)
      with Sys_error e ->
        Error.f (s_ "Cannot save to %s\n(%s)" f e)
     end;
     close_out oc
    with Sys_error e ->
      Error.f (s_ "Cannot save to %s\n(%s)" f e)
```

⟨*function* Save.copy_file 337a⟩≡ (339b)

```
  let copy_file url src dst =
    try
      let ic = open_in_bin src
      and oc = open_out_bin dst
      and buf = Bytes.create 2048 in
      let rec copy () =
        let n = input ic buf 0 2048 in
        if n <> 0 then begin output oc buf 0 n; copy() end
      in
      begin try
        copy();
        Error.ok (s_ "Document %s\nsaved in\n%s" (Url.string_of url) dst)
      with Sys_error e ->
        Error.f (s_ "Cannot save to %s\n(%s)" dst e)
      end;
      close_in ic;
      close_out oc
    with Sys_error e ->
      Error.f (s_ "Cannot save to %s\n(%s)" dst e)
```

⟨*function* Save.pipe_from_string 337b⟩≡ (339b)

```
  (* Cmd can be composite. We add the URL at the end *)
  let pipe_from_string url data cmd =
    let urls = Url.string_of url in
    try
      (* we have to open a pipe and write to it *)
      let fd_in, fd_out = pipe() in
      let len = String.length data and pos = ref 0 in
      (* now fork the command *)
      match Low.fork() with
      (* child *)
      | 0 ->
        dup2 fd_in stdin; close fd_in; close fd_out;
        ignore (Munix.system_eval cmd ["URL", urls] false);
        (* nosemgrep: do-not-use-exit *)
        exit 0
      (* parent *)
      | _n ->
         close fd_in;
         Fileevent.add_fileoutput fd_out (fun () ->
        if !pos < len then begin
          let n = min 512 (len - !pos) in
          try
            let w = write fd_out (Bytes.of_string data) !pos n in
            pos := !pos + w
          with
            Unix_error (_,_,_) -> (* can't write *)
          Fileevent.remove_fileoutput fd_out;
          close fd_out;
          Error.f (s_ "Error during |%s in %s" cmd urls)
        end else begin (* we're done *)
          Fileevent.remove_fileoutput fd_out;
          close fd_out
        end)
    with
    | Unix_error(_,_,_) -> (* pipe failed, fork failed *)
        Error.f (s_ "Can't execute command %s for %s" cmd urls)
```

⟨*function* Save.pipe_from_file 338a⟩≡ (339b)

```
 let pipe_from_file url f cmd =
   let urls = Url.string_of url in
   try
     (* just open the file and read from it *)
     match Low.fork() with
     (* child *)
     | 0 ->
       let fd = openfile f [O_RDONLY] 0 in
       dup2 fd stdin; close fd;
       ignore (Munix.system_eval cmd ["URL", urls] false);
       (* nosemgrep: do-not-use-exit *)
       exit 0
      (* parent *)
      | _n ->
        ()
   with Unix_error(_,_,_) -> (* pipe failed, fork failed *)
     Error.f (s_ "Can't execute command %s for %s" cmd urls)
```

⟨*function* Save.document 338b⟩≡ (339b)

```
 let document did arg =
   let open_selection_box act =
     Fileselect.f (s_ "Save or pipe to file")
       (function [] -> ()
              | [s] -> act s
           | _l -> raise (Failure "multiple selection"))
       "*" (* should be better *)
       (Filename.basename (Url.string_of did.document_url))
       false
       true
   in
   let proceed f = match arg with
     None -> open_selection_box f
   | Some s -> f s
   in
   try
     match Cache.find did with
       {document_data = MemoryData buf; _} ->
         proceed
       (fun s ->
         if String.length s <> 0 && s.[0] == '|' then
           pipe_from_string did.document_url (Ebuffer.get buf)
         (String.sub s 1 (String.length s - 1))
         else
           save_from_string did.document_url (Ebuffer.get buf) s)

     |  {document_data = FileData (f, _); _} ->
         proceed
       (fun s ->
         if String.length s <> 0 && s.[0] == '|' then
           pipe_from_file did.document_url (Fpath.to_string f)
         (String.sub s 1 (String.length s - 1))
         else
```

```
          copy_file did.document_url (Fpath.to_string f) s)
    with Not_found ->
      Error.f ("Document is not in cache.")
```

⟨*constant* Save.print_command 339a⟩≡ (339b)

```
  let print_command = ref ""
```

⟨viewers/save.ml 339b⟩≡

```
  open I18n

  open Unix
  open Document
  open Url
  open Www
  open Feed

  ⟨function Save.f 335e⟩

  ⟨function Save.tofile 336a⟩

  ⟨function Save.interactive 121e⟩


  ⟨function Save.transfer 336b⟩

  ⟨function Save.save_from_string 336c⟩

  ⟨function Save.copy_file 337a⟩


  ⟨function Save.pipe_from_string 337b⟩
  ⟨function Save.pipe_from_file 338a⟩

  ⟨function Save.document 338b⟩

  ⟨constant Save.print_command 339a⟩
```

F.7.7 viewers/viewers.mli

⟨viewers/viewers.mli 339c⟩≡

```

  ⟨type Viewers.vparams 163e⟩
  ⟨type Viewers.frame_targets 158e⟩

  ⟨type Viewers.hyper_func 24d⟩

  ⟨signature class Viewers.context 24a⟩

  ⟨signature class Viewers.display_info 25a⟩


  ⟨signature Viewers.di_compare 237e⟩

  ⟨type Viewers.t 23f⟩

  ⟨signature Viewers.add_viewer 111c⟩
  ⟨signature Viewers.rem_viewer 111e⟩
  ⟨signature Viewers.add_builtin 112c⟩
  ⟨signature Viewers.reset 112e⟩
  (* !!! main entry point!!!! *)
  ⟨signature Viewers.view 111a⟩

  ⟨signature Viewers.frame_adopt 158f⟩
  ⟨signature Viewers.frame_fugue 159a⟩
```

F.7.8 viewers/viewers.ml

```
⟨function Viewers.metamail 340a⟩≡                                              (340c)
  (* Metamail options
     -b : not an RFC822 message
     -z : delete when finished
     -x : not on a tty
   *)
  let metamail ctype file =
    ignore (Munix.system "metamail -b -z -x -c" [ctype; file] true)

⟨function Viewers.extern_batch 340b⟩≡                                          (340c)
  (* Batch version: we transfer everything and then run metamail *)
  let _extern_batch dh ctype =
    let outfile = Msys.mktemp "mmm" in
    Document.add_log dh (
      s_ "Saving %s\nfor external display with MIME type %s"
           (Url.string_of dh.document_id.document_url) ctype)
      (fun () -> Msys.rm outfile);
    let endmsg =
      s_ "Running metamail with MIME media-type: %s" ctype in
      Save.tofile (metamail ctype) (Decoders.insert dh) outfile endmsg

⟨viewers/viewers.ml 340c⟩≡
  open Common
  open I18n

  (*****************************************************************************)
  (* Prelude *)
  (*****************************************************************************)
  (*
   * Multimedia
   *)

  (*****************************************************************************)
  (* Types *)
  (*****************************************************************************)

  ⟨type Viewers.vparams 163e⟩
  ⟨type Viewers.frame_targets 158e⟩

  ⟨function Viewers.frame_adopt 159b⟩

  ⟨function Viewers.frame_fugue 159c⟩

  ⟨type Viewers.hyper_func 24d⟩

  ⟨class Viewers.context 24e⟩

  (* The object created/returned by a viewer *)
  class  virtual display_info () =
   object (_self : 'a)
    (* boilerplate class decl *)
    ⟨Viewers.display_info virtual methods signatures 25b⟩
    ⟨Viewers.display_info graphic cache methods 237d⟩
  end

  (*****************************************************************************)
  (* Helpers *)
  (*****************************************************************************)

  ⟨function Viewers.di_compare 237f⟩

  (*****************************************************************************)
  (* External viewer *)
  (*****************************************************************************)
  (*
   * The default external viewer
   *)

  ⟨function Viewers.metamail 340a⟩

  ⟨function Viewers.extern_batch 340b⟩

  ⟨function Viewers.extern 119b⟩

  (*****************************************************************************)
  (* Types part 2 *)
  (*****************************************************************************)

  (*
   * Viewer control
   * Specify on the base of MIME type if we want to
   *   - use an internal displayer (assumed to exist)
   *   - use an external displayer (metamail or other)
   *   - save to file
   *)

  (* Table of viewers, according to media-type (MIME)
   * Actually, this is only for internal viewers, since the rest
   * will be passed to metamail.
   *)

  ⟨type Viewers.t 23f⟩

  ⟨type Viewers.spec 23h⟩

  ⟨constant Viewers.viewers 23g⟩

  ⟨function Viewers.add_viewer 111d⟩

  ⟨function Viewers.rem_viewer 112a⟩

  (*****************************************************************************)
  (* Interactive viewer *)
  (*****************************************************************************)
  ⟨function Viewers.unknown 113a⟩

  ⟨function Viewers.interactive 120d⟩

  (*****************************************************************************)
  (* Main entry point *)
```

```
  (*****************************************************************************)
  ⟨function Viewers.view 111b⟩

  (*****************************************************************************)
  (* Builtin viewers global *)
  (*****************************************************************************)
  ⟨constant Viewers.builtin_viewers 112b⟩
  ⟨function Viewers.add_builtin 112d⟩

  ⟨function Viewers.reset 112f⟩
```

F.7.9 viewers/plain.ml

```
⟨viewers/plain.ml 342a⟩≡
  open Tk

  (*****************************************************************************)
  (* Prelude *)
  (*****************************************************************************)

  (*****************************************************************************)
  (* Display_info for "text/plain" *)
  (*****************************************************************************)

  ⟨class Plain.plain 113d⟩

  (*****************************************************************************)
  (* Entry point *)
  (*****************************************************************************)
  ⟨function Plain.display_plain 113c⟩

  ⟨toplevel Plain._1 113b⟩
```

F.8 display/

F.8.1 display/attrs.mli

```
⟨signature Attrs.color_mappings 342b⟩≡                                    (343)
  val color_mappings : (string, string) Hashtbl.t

⟨signature Attrs.html_color 342c⟩≡                                        (343)
  val html_color : string -> string

⟨signature Attrs.circle_data 342d⟩≡                                       (343)
  val circle_data : string

⟨signature Attrs.disc_data 342e⟩≡                                         (343)
  val disc_data : string

⟨signature Attrs.square_data 342f⟩≡                                       (343)
  val square_data : string

⟨signature Attrs.bullet_table 342g⟩≡                                      (343)
  val bullet_table : (string, Tk.options) Hashtbl.t

⟨signature Attrs.init 342h⟩≡                                              (343)
  val init : string -> unit
```

⟨display/attrs.mli 343⟩≡
```
  module TagSet : Set.S with type elt = string

  class tags :
    Widget.widget ->
    object
      val mutable configured : TagSet.t
      val mutable decorations : (Tk.textTag * Tk.textIndex * Tk.textIndex) list
      val mutable onhold : (TagSet.elt * Tk.options list) list
      val wid : Widget.widget
      method add : Tk.textTag * Tk.textIndex * Tk.textIndex -> unit
      method change : TagSet.elt -> Tk.options list -> unit
      method define : TagSet.elt -> Tk.options list -> unit
      method flush : unit
    end

  module LocMap :
    sig
      type key = Tk.index
      type 'a t
      val empty : 'a t
      val add : key * key -> 'a -> 'a t -> 'a t
      val find : key -> 'a t -> 'a
      val find_interval : key -> 'a t -> key * key
    end

  class anchortags :
    Widget.widget ->
    object
      val mutable anchor_table : Hyper.link LocMap.t
      val mutable configured : TagSet.t
      val mutable decorations : (Tk.textTag * Tk.textIndex * Tk.textIndex) list
      val mutable mappings : (Tk.textIndex * Tk.textIndex * Hyper.link) list
      val mutable onhold : (TagSet.elt * Tk.options list) list
      val wid : Widget.widget
      method add : Tk.textTag * Tk.textIndex * Tk.textIndex -> unit
      method add_anchor : Tk.textIndex -> Tk.textIndex -> Hyper.link -> unit
      method binder :
        (Tk.modifier list * Tk.xEvent) list -> Tk.bindAction -> unit
      method change : TagSet.elt -> Tk.options list -> unit
      method define : TagSet.elt -> Tk.options list -> unit
      method flush : unit
      method getlink : Tk.eventInfo -> Hyper.link
      method getrange : LocMap.key -> LocMap.key * LocMap.key
      method highlight : bool -> unit
      method init : Viewers.context -> unit
      method markused : Tk.eventInfo -> unit
      method widget : Widget.widget
    end

  class virtual ['a] nested :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> 'a -> unit
      method virtual pop_convert : 'a -> unit
      method push : Tk.textIndex -> 'a -> unit
      method virtual push_convert : 'a -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
    end

  class align :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> string -> unit
      method pop_convert : string -> unit
      method push : Tk.textIndex -> string -> unit
      method push_convert : string -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
    end

  class margin :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable current : int
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> int -> unit
      method pop_convert : int -> unit
      method push : Tk.textIndex -> int -> unit
      method push_convert : int -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
    end

  class font :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable basefont : Fonts.fontDesc
      val mutable font_stack : Fonts.fontDesc list
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> Fonts.fontAttrs -> unit
      method pop_all : Tk.textIndex -> unit
      method pop_convert : Fonts.fontAttrs -> unit
      method push : Tk.textIndex -> Fonts.fontAttrs -> unit
      method push_convert : Fonts.fontAttrs -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
      method set_base : Tk.textIndex -> int -> unit
    end

  ⟨signature Attrs.color_mappings 342b⟩
  ⟨signature Attrs.html_color 342c⟩

  class fgcolor :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> string -> unit
      method pop_convert : string -> unit
      method push : Tk.textIndex -> string -> unit
      method push_convert : string -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
    end

  class bgcolor :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> string -> unit
      method pop_convert : string -> unit
      method push : Tk.textIndex -> string -> unit
      method push_convert : string -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
    end

  class offset :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      val mutable cur_offset : int
      val mutable last_change : Tk.textIndex
      val mutable stack : string list
      method pop : Tk.textIndex -> int -> unit
      method pop_convert : int -> unit
      method push : Tk.textIndex -> int -> unit
      method push_convert : int -> string * Tk.options list
      method put : Tk.textIndex -> string -> unit
    end

  class misc :
    < add : 'a * Tk.textIndex * Tk.textIndex -> unit;
      define : 'a -> 'b -> 'c;
      .. > *
    'a * 'b ->
    object
      val mutable start_pos : Tk.textIndex
      val tagname : 'a
      method pop : Tk.textIndex -> unit
      method push : Tk.textIndex -> unit
    end

  class spacing :
    < add : string * Tk.textIndex * Tk.textIndex -> unit;
      define : string -> Tk.options list -> unit; .. > ->
    object
      method pop : Tk.textIndex -> int -> unit
      method push : Tk.textIndex -> int -> unit
    end
```

⟨*signature* `Attrs.circle_data` 342d⟩
⟨*signature* `Attrs.disc_data` 342e⟩
⟨*signature* `Attrs.square_data` 342f⟩
⟨*signature* `Attrs.bullet_table` 342g⟩
⟨*signature* `Attrs.init` 342h⟩

F.8.2 display/attrs.ml

⟨*constant* `Attrs.color_mappings` 345⟩≡ (347b)

```
  (* Special mapping of pre-defined HTML3.2 colors *)
  let color_mappings = Hashtbl.create 37
```

⟨*toplevel* Attrs._1 346a⟩≡ (347b)

```
  let _ = List.iter (fun (name, value) -> Hashtbl.add color_mappings name value)
    [ "black",   "#000000";
      "silver",  "#c0c0c0";
      "gray",    "#808080";
      "white",   "#ffffff";
      "maroon",  "#800000";
      "red",     "#ff0000";
      "purple",  "#800080";
      "fuchsia", "#ff00ff";
      "green",   "#008000";
      "lime",    "#00ff00";
      "olive",   "#808000";
      "yellow",  "#ffff00";
      "navy",    "#000080";
      "blue",    "#0000ff";
      "teal",    "#008080";
      "aqua",    "#00ffff" ]
```

⟨*function* Attrs.html_color 346b⟩≡ (347b)

```
  let html_color s =
    try Hashtbl.find color_mappings (String.lowercase_ascii s)
    with Not_found -> s
```

⟨*constant* Attrs.circle_data 346c⟩≡ (347b)

```
  (* Bullet images *)
  let circle_data =
  "#define circle_width 9
  #define circle_height 9
  static unsigned char circle_bits[] = {
     0x00, 0x00, 0x00, 0x00, 0x38, 0x00, 0x44, 0x00, 0x44, 0x00, 0x44, 0x00,
     0x38, 0x00, 0x00, 0x00, 0x00, 0x00};"
```

⟨*constant* Attrs.disc_data 346d⟩≡ (347b)

```
  let disc_data =
  "#define disc_width 9
  #define disc_height 9
  static unsigned char disc_bits[] = {
     0x00, 0x00, 0x00, 0x00, 0x38, 0x00, 0x7c, 0x00, 0x7c, 0x00, 0x7c, 0x00,
     0x38, 0x00, 0x00, 0x00, 0x00, 0x00};"
```

⟨*constant* Attrs.square_data 346e⟩≡ (347b)

```
  let square_data =
  "#define square_width 9
  #define square_height 9
  static unsigned char square_bits[] = {
     0x00, 0x00, 0x00, 0x00, 0x7c, 0x00, 0x7c, 0x00, 0x7c, 0x00, 0x7c, 0x00,
     0x7c, 0x00, 0x00, 0x00, 0x00, 0x00};"
```

⟨*constant* Attrs.bullet_table 346f⟩≡ (347b)

```
  let bullet_table = Hashtbl.create 11
```

⟨*function* Attrs.init 347a⟩≡ (347b)
```
  let init bg =
    let _bgTODO = Background (NamedColor bg) in
    Hashtbl.add bullet_table
       "circle" (ImageBitmap(Imagebitmap.create [Data circle_data]));
    Hashtbl.add bullet_table
       "disc" (ImageBitmap(Imagebitmap.create [Data disc_data]));
    Hashtbl.add bullet_table
       "square" (ImageBitmap(Imagebitmap.create [Data square_data]))
```

⟨display/attrs.ml 347b⟩≡
```

  (* Utilities for tags and attributes *)

  open Printf
  open Protocol
  open Tk
  open Fonts


  (* Delayed and shared configuration of tags *)

  module TagSet = Set.Make(struct type t = string let compare = compare end)

  class tags (thtml) =
   object
    val mutable onhold = []
    val mutable configured = TagSet.empty
    val mutable decorations = []
    val wid = thtml

    (* define a new tag *)
    method define tagname attrs =
      if TagSet.mem tagname configured then ()
      else begin
        onhold <- (tagname,attrs) :: onhold;
        configured <- TagSet.add tagname configured
      end

    (* change a tag value *)
    method change tagname attrs =
      onhold <- (tagname,attrs) :: onhold;
      configured <- TagSet.add tagname configured

    method add deco =
      decorations <- deco :: decorations

    (* flush tag definitions *)
    method flush =
      onhold |> List.rev |> List.iter (fun (t,d) ->
        try Text.tag_configure wid t d with TkError _ -> ());
      decorations |> List.rev |> List.iter (fun (t,d,e) ->
        Text.tag_add wid t d e);
      onhold <- [];
      decorations <- []
  end

  module LocMap = Ibtree.Make(struct
    type t = index
    let compare = compare
    end)

  class anchortags (thtml) =
   object (_self)
    inherit tags (thtml) as tags
    inherit Htbind.hypertext (thtml)

    val mutable mappings = []
    val mutable anchor_table = LocMap.empty

    method add_anchor s e h =
      tags#add ("anchor", s, e);
      mappings <- (s,e,h) :: mappings

    method! flush =
      tags#flush;
      mappings |> List.iter (fun (s,e,h) ->
        let loc1 = Text.index wid s
        and loc2 = Text.index wid e
        in
        anchor_table <- LocMap.add (loc1,loc2) h anchor_table
      );
      mappings <- []

    method getlink ei =
       (* The index of the click position *)
       let i =
         Text.index thtml (TextIndex (AtXY (ei.ev_MouseX,ei.ev_MouseY), [])) in
       LocMap.find i anchor_table

    method getrange i = LocMap.find_interval i anchor_table

  end


  (* Conversion of moral attributes to Tk attributes.
   * This virtual class has to be instantiated for each converter.
   * 'a is an logical attribute description (or "delta")
   *)
  class  virtual ['a] nested (tagdef) =
   object (self)
    val mutable last_change = TextIndex(LineChar(0,0),[])
    val mutable stack = []
    (* val tagdef = tagdef *)

    method virtual push_convert : 'a -> string * Tk.options list
    method virtual pop_convert : 'a -> unit

    method put current_pos tagname =
     if last_change <> current_pos then begin
       tagdef#add (tagname, last_change, current_pos);
       last_change <- current_pos
     end

    (* Push some new attribute. *)
    method push current_pos desc =
      let tag, attr = self#push_convert desc in
      tagdef#define tag attr;
      begin match stack with
         [] ->
```

```
        (* no current definition, don't issue a put *)
        last_change <- current_pos
     | curtag::_l ->
        self#put current_pos curtag
    end;
    stack <- tag :: stack;

(* Doesn't check the nature of desc *)
  method pop current_pos (desc : 'a) =
     self#pop_convert desc;
     match stack with
       [] ->
        last_change <- current_pos
     | c::l ->
    stack <- l;
        self#put current_pos c
end

(*
 * Alignment attribute is left/right/center
 *)
class align (tagdef) =
 object
  inherit [string] nested tagdef
  method push_convert ad =
    match String.lowercase_ascii ad with
     "right" -> "right", [Justify Justify_Right]
       | "center" -> "center", [Justify Justify_Center]
       | _ -> "left", [Justify Justify_Left]
  method pop_convert _ad = ()
end

(*
 * Margin attribute is cumulative
 *)
class margin (tagdef) =
 object
  inherit [int] nested tagdef
  val mutable current = 0
  method push_convert ad =
    current <- current + ad;
    sprintf "margin%d" current,
    [LMargin1 (Pixels current); LMargin2 (Pixels current)]
  method pop_convert ad =
    current <- current - ad
end


(*
 * Font attributes
 *)

class font (tagdef) =
 object (self)
  inherit [fontInfo list] nested tagdef
  val mutable font_stack = []
  val mutable basefont = !Fonts.default
  method push_convert fil =
    let curfd = match font_stack with
      [] -> basefont
    | x::_l -> x in
    let newfd = Fonts.merge curfd fil in
      font_stack <- newfd :: font_stack;
      Fonts.compute_tag newfd

  method pop_convert _ =
    match font_stack with
      [] -> ()
    | _x::l -> font_stack <- l

  (* by changing the base, we should be changing both the current default size
     and the behaviour of subsequent FONT SIZE tags. The size changes is easy.
     The header styles being defined with an absolute font, they are not
     affected
     It's logical also to push this as the current font, but the problem
     is that it doesn't work because basefont do not obey nesting rules
     (consider <FONT> <BASEFONT> </FONT> !). We do deal with this situation.
   *)
  method set_base current_pos n =
    basefont <- { basefont with pxlsz= n };
    self#push current_pos [FontIndex n];

  method pop_all current_pos =
    while font_stack <> [] do
      self#pop current_pos []
    done

end


⟨constant Attrs.color_mappings 345⟩
⟨toplevel Attrs._1 346a⟩

⟨function Attrs.html_color 346b⟩

(*
 * Foreground color
 *)

class fgcolor (tagdef) =
 object
  inherit [string] nested tagdef
  method push_convert s =
    let colordef = html_color s in
    if Frx_color.check colordef then
      s, [Foreground (NamedColor colordef)]
    else
      s, []
  method pop_convert _s =
    ()
end

(*
 * Background color
 *)

class bgcolor (tagdef) =
 object
  inherit [string] nested tagdef
  method push_convert s =
    let colordef = html_color s in
    if Frx_color.check colordef then
      s, [Background (NamedColor colordef)]
    else
      s, []
  method pop_convert _s =
    ()
end

(*
 * Super and sub script.
 * BOGUS: should depend on current font size
 *)
class offset (tagdef) =
 object
  inherit [int] nested tagdef
  val mutable cur_offset = 0
  method push_convert n =
    cur_offset <- cur_offset + n;
    sprintf "offset%d" cur_offset, [Offset (Pixels cur_offset)]
  method pop_convert n =
    cur_offset <- cur_offset - n
end
(*
 * Other stuff where nesting is not important
*)
class misc (tagdef, tagname, attr) =
 object (_self)

  val mutable start_pos = TextIndex(LineChar(0,0),[])
  (* val tagdef = tagdef *)
  val tagname =
     let _ = tagdef#define tagname  attr in
     tagname

  method pop current_pos =
   if start_pos <> current_pos then begin
     tagdef#add (tagname, start_pos, current_pos)
   end

  method push current_pos  =
     start_pos <- current_pos

end

(*
 * Spacing is specific, due to Tk's line model and BR
 *  push corresponds to top spacing for the first line
 *  pop corresponds to bottom spacing for the first line
 *)

class spacing (tagdef) =
 object
  (* val tagdef = tagdef *)

  method push current_pos n =
    let topname = sprintf "topspace%d" n in
     tagdef#define topname [Spacing1 (Pixels n)];
     match current_pos with
       TextIndex(base,[x]) ->
     tagdef#add (topname, TextIndex(base, [x;LineStart]),
                     TextIndex(base, [x;LineEnd]));
     ()
     | _ -> assert false

  method pop current_pos n =
    let botname = sprintf "botspace%d" n in
    tagdef#define botname [Spacing3 (Pixels n)];
     match current_pos with
       TextIndex(base,[x]) ->
     tagdef#add (botname, TextIndex(base, [x;LineStart]),
                     TextIndex(base, [x;LineEnd]));
     ()
     | _ -> assert false
end

⟨constant Attrs.circle_data 346c⟩

⟨constant Attrs.disc_data 346d⟩

⟨constant Attrs.square_data 346e⟩

⟨constant Attrs.bullet_table 346f⟩
⟨function Attrs.init 347a⟩
```

F.8.3 display/cmap.ml

⟨*exception* Cmap.Syntax 352a⟩≡ (354a)

```
  (* Client Side Image Maps
     We must have two modes: one when the image has not been loaded.
     In that case, we need something like a popup menu. And then, when
     the image is loaded, we use a canvas
     *)

  exception Syntax of string
```

⟨*function* Cmap.alt_mode 352b⟩≡ (354a)

```
  let alt_mode emb m l =
    Log.debug (sprintf "Alt mode map for %s" (Widget.name l));
    let menu = Menu.create_named l "map" [] in
    List.iter (fun area ->
      Menu.add_command menu
        [Label (if area.area_alt = "" then
      area.area_link.h_uri
        else area.area_alt);
      Command (fun () -> emb.embed_context#goto area.area_link)])
      m;
    bind l (Glevents.get "alt_imap")
      (BindSet ([Ev_RootX; Ev_RootY],
            (fun ei -> Menu.popup menu ei.ev_RootX ei.ev_RootY)))
```

⟨*function* Cmap.printTagOrId 352c⟩≡ (354a)

```
  let printTagOrId = function
    | Id n -> Log.f (sprintf "Id %d" n)
    | Tag s -> Log.f (sprintf "Tag %s" s)
```

⟨*function* Cmap.gfx_mode 353⟩≡ (354a)

```
  (* This is called when the image has been loaded *)
  let gfx_mode emb map c =
    Log.debug (sprintf "Gfx mode map for %s" (Widget.name c));

    (* Build the canvas items corresponding to active zones *)

    (* For points *inside* rects and circle items to be actually considered
       inside for the purpose of activation, we must use both an empty outline
       and an empty fill. *)
    let opts = [Outline (NamedColor ""); FillColor (NamedColor "")] in

    let items =
      List.map (fun area ->
        try
          match area.area_kind with
        Default -> Id 1, area.area_link (* the image itself *)
          | Rect ->
          begin match area.area_coords with
          | [x1;y1;x2;y2] ->
             Canvas.create_rectangle c
            (Pixels x1) (Pixels y1) (Pixels x2) (Pixels y2)
            opts,
             area.area_link
          | _ ->
          raise (Syntax "rect")
          end
          | Circle ->
          begin match area.area_coords with
          | [x;y;r] ->
             Canvas.create_oval c
            (Pixels (x-r)) (Pixels (y-r))
            (Pixels (x+r)) (Pixels (y+r))
            opts,
             area.area_link
          | _ -> raise (Syntax "circle")
          end
          | Poly ->
          let l = List.length area.area_coords in
          (* there must be at least three points, and by pair *)
          if l < 6 || l mod 2 <> 0 then begin
            Log.f "Invalid coords for polygon shape";
            raise (Syntax "polygon")
          end
          else
            Canvas.create_polygon c
                 (List.map (fun x -> Pixels x) area.area_coords)
                 opts,
            area.area_link
        with
        | Syntax s ->
        Log.f (sprintf "Wrong syntax in area mapping (%s)" s);
        Tag "area error", area.area_link
        | Protocol.TkError s ->
        Log.f (sprintf "Error in area mapping (%s)" s);
        Tag "area error", area.area_link
       )
       map
    in
    Canvas.lower_bot c (Id 1);
    let htobj = new imap(c,items) in
      htobj#init emb.embed_context
```

⟨display/cmap.ml 354a⟩≡

```
  open Printf
  open Tk
  open Embed
  open Maps
  open Hyper


  ⟨exception Cmap.Syntax 352a⟩

  ⟨function Cmap.alt_mode 352b⟩

  ⟨function Cmap.printTagOrId 352c⟩


  (* See Htbind for semantics of this class *)
  class imap (c, items) =
   object (self)
   inherit Htbind.active () as super

   (* val items = items *)
   (* val c = c *)
   method widget = c

   method getlink ei =
     let cx = truncate (Canvas.canvasx c (Pixels ei.ev_MouseX))
     and cy = truncate (Canvas.canvasy c (Pixels ei.ev_MouseY)) in
       match Canvas.find c [Closest (Pixels cx, Pixels cy)] with
         [id] -> List.assoc id items
       |  _ -> raise Not_found

   method binder = Canvas.bind c (Tag "current")

   method highlight _ = ()
   method markused _ei = ()

   method! init ctx =
     super#init ctx;
     self#binder [[], Motion]
      (BindSet ([Ev_MouseX; Ev_MouseY],
            (fun ei ->
              try
                let link = self#getlink ei in
                ctx#invoke "pointsto" link
              with
                Not_found -> ())))
  end

  ⟨function Cmap.gfx_mode 353⟩
```

F.8.4 display/fit.mli

⟨*signature* Fit.debug 354b⟩≡ (355f)

```
  val debug : bool ref
```

⟨*signature* Fit.set_initial_width 354c⟩≡ (355f)

```
  val set_initial_width : Widget.widget -> int option
```

⟨*signature* Fit.set_initial_height 355a⟩≡ (355f)
```
  val set_initial_height: Widget.widget -> unit
```

⟨*signature* Fit.horiz 355b⟩≡ (355f)
```
  (* [horiz textw stop continuation] returns [scrollcommand, check] *)
  val horiz:
    Widget.widget -> (unit -> bool) -> (unit -> unit) ->
    (float -> float -> unit) * (unit -> unit)
```

⟨*signature* Fit.vert 355c⟩≡ (355f)
```
  val vert:
    Widget.widget ->
    (float -> float -> unit) * (unit -> unit)
```

⟨*signature* Fit.bound_check 355d⟩≡ (355f)
```
  val bound_check : Widget.widget -> int -> (unit -> bool)
```

⟨*signature* Fit.fixed_horiz 355e⟩≡ (355f)
```
  val fixed_horiz : Widget.widget -> int -> unit
```

⟨display/fit.mli 355f⟩≡

  ⟨*signature* Fit.debug 354b⟩

  ⟨*signature* Fit.set_initial_width 354c⟩
  ⟨*signature* Fit.set_initial_height 355a⟩

  ⟨*signature* Fit.horiz 355b⟩

  ⟨*signature* Fit.vert 355c⟩

  ⟨*signature* Fit.bound_check 355d⟩

  ⟨*signature* Fit.fixed_horiz 355e⟩

F.8.5 display/fit.ml

⟨*constant* Fit.debug 355g⟩≡ (359b)
```
  let debug = ref false
```

⟨*function* Fit.set_initial_width 355h⟩≡ (359b)
```
  (* initial width : a nested formatter starts with w=1 h=1
   * if the only contents is an embedded window, it's a bit short
   * thus we check for max width of embedded windows
   * We take an arbitrary width of 10 pixels per char.
   *)
  let set_initial_width wid =
    let ewidth = ref 0 in
    Text.window_names wid |> List.iter (fun w ->
      ewidth := max (Winfo.reqwidth w) !ewidth
    );
    if !ewidth > 10 then begin
      let w = !ewidth / 10 in
      if !debug
      then Log.f (sprintf "Setting initial width of %s to %d"(Widget.name wid) w);
      Text.configure wid [TextWidth w];
      Some w
    end
    else None
```

⟨*function* Fit.wheight 356a⟩≡ (359b)
```
  (* I still don't understand the difference between height and reqheight *)
  let wheight w = max (Winfo.height w) (Winfo.reqheight w)
```

⟨*function* Fit.set_initial_height 356b⟩≡ (359b)
```
  let set_initial_height wid =
    match Text.index wid (TextIndex(End,[])) with
      LineChar (l,_) ->
        let height = ref (l-1) in
        for i = 0 to l - 1 do
      match Text.index wid (TextIndex(LineChar(i,0), [LineEnd])) with
        LineChar (_,c) -> height := !height + c / 100
      | _ -> ()
        done;
        begin
      let embedded = Text.window_names wid in
      if embedded = [] then ()
      else begin
        let lines = Hashtbl.create (List.length embedded) in
        let addh l h =
          try
            let r = Hashtbl.find lines l in
            r := max h !r
          with
            Not_found -> Hashtbl.add lines l (ref h)
        in
        List.iter (fun w ->
          match Text.index wid (TextIndex(Embedded w, [])) with
            LineChar(l,_) -> addh l (wheight w)
          | _ -> assert false)
          embedded;
        Hashtbl.iter (fun _ r -> height := !height + !r / 15) lines
         end;
       let curheight = int_of_string (cget wid CHeight) in
      if !height > curheight then begin
        if !debug then
          Log.f (sprintf "Setting initial height of %s to %d (%d)"
                       (Widget.name wid) !height (l-1));
           Text.configure wid [TextHeight (!height)]
      end else if !debug then
          Log.f (sprintf "Initial height of %s is %d (%d)"
                       (Widget.name wid) !height (l-1))
        end
    | _ -> ()
```

⟨*function* Fit.fixed_horiz 356c⟩≡ (359b)
```
  (* Grow horizontally until we reached the maxium authorized width
   * (or bound is reached)
   *)
  let rec fixed_horiz wid maxw =
    let curw = Winfo.reqwidth wid in
    if  curw >= maxw then ()
    else begin
      let w = (succ (int_of_string (cget wid CWidth))) in
      if !debug then
        Log.f (sprintf "Growing %s to %d (w=%d) (max=%d)"
                    (Widget.name wid) w curw maxw);
      Text.configure wid [TextWidth w];
      fixed_horiz wid maxw
    end
```

⟨*function* Fit.horiz 357⟩≡ (359b)
```
  let horiz wid stop continuation =
    (* all conditions for stopping *)
    let finished visible = visible >= 0.999 || stop()
    (* bail out *)
    and over () =
     (* deconnect ourselves *)
     (*Text.configure wid [XScrollCommand (fun _ _ -> ())];*)
     continuation()
    and last_visible = ref 0.0
    in
    (* if we want to restart after we were disconnected*)
    let rec check () =
      let first, last = Text.xview_get wid in
       scroll first last
    (* binding to XScrollCommand *)
    and scroll first last =
      let curwidth = int_of_string (cget wid CWidth)
      and visible = last -. first in
      (* Don't attempt anything if widget is not visible *)
      (* Especially, DO NOT DECIDE TO STOP *)
      if not (Winfo.viewable wid) then begin
        if !debug then
      Log.f (sprintf "%s HC %d %f %f notviewable"
                     (Widget.name wid) curwidth first last);
        (* Try again later *)
        bind wid [[], Expose] (BindSet ([], fun _ ->
      bind wid [[], Expose] BindRemove;
      check()))
      end
      else if finished visible then over()
      else if visible = !last_visible then
        (* it didn't change since our last resize ! *)
        ()
      else begin
        (* how much do we need to grow
       This code is disabled because it causes masking of table cells
       (we don't have a reasonable estimation of a minimum horiz growth
        that would avoid masking). We now grow by 1, despite the
        slowness.
        let delta =
      if last = 0.0 then 1
          else begin
        last_visible := visible;
        let visible = max 0.2 visible in
        let missing = 1. -. visible in
        (* at least one char, but not too much *)
        let computed = truncate (float curwidth *. missing /. visible) in
        if computed = 0 then 1 else min 5 computed
          end
        in
        let newsize = curwidth + delta in
       *)
        last_visible := visible;
        let newsize = curwidth + (if visible < 0.1 then 5 else 1) in
        if !debug then
         Log.f (sprintf "%s H %d %f %f newsize: %d"
                     (Widget.name wid) curwidth first last newsize);
        Text.configure wid [TextWidth newsize];
      end
    in
    scroll, check
```

⟨*function* Fit.vert 358⟩≡ (359b)

```
  (* somehow we need to do it differently : resize is delayed *)
  let vert wid =
    let finished visible = visible >= 0.999
    and last_visible = ref (-1.0)   (* last value of visible *)
    and stuck = ref false     (* last resize didn't have an effect *)
    and pending_check = ref false
    and delayed = ref false   (* we have a binding on Expose *)
    and pending_resize = ref false (* a resize is pending *)
    and newsize = ref 0
    in
    let rec check () =
      if Winfo.exists wid then begin (* we must check since we use a delay *)
      let first, last = Text.yview_get wid in
        pending_check := false;
        scroll first last
      end
    and scroll first last =
      let curheight = int_of_string (cget wid CHeight)
      and visible = last -. first in
      (* Don't attempt anything if widget is not visible *)
      if not (Winfo.viewable wid) then begin
        if !debug then
      Log.f (sprintf "%s VC %d %f %f notviewable"
                     (Widget.name wid) curheight first last);
        (* Try again later *)
        if not !delayed then begin
      delayed := true;
         bind wid [[], Expose] (BindSet ([], fun _ ->
        bind wid [[], Expose] BindRemove;
        delayed := false;
        check()))
        end
      end
      else if finished visible then ()
      else if !stuck then
        if !pending_check then ()
        else begin
        (* last check had same last value than before *)
        pending_check := true;
        if !debug then Log.f (sprintf "Stuck %s" (Widget.name wid));
        Timer.set 50 (fun () -> stuck := false; Frx_after.idle check)
        end
      else begin
        let delta =
    if visible = !last_visible then (stuck := true; 1)
          else if last = 0.0 then (last_visible := 0.0; 1)
      else begin
        last_visible := visible; stuck := false;
            (* never to more than double *)
        let visible = max 0.5 visible in
        let missing = 1. -. visible in
        (* at least one char, but not too much *)
        let computed = truncate (float curheight *. missing /. visible) in
        if computed = 0 then 1 else min 5 computed
          end
        in
        newsize := max (curheight + delta) !newsize;
        (* Since we may not be fully visible anyway, decouple the loop *)
        if !pending_resize then ()
        else begin
      if !debug then
        Log.f (sprintf "Scheduling resize of %s" (Widget.name wid));
      pending_resize := true;
          Timer.set 50 (fun () -> Frx_after.idle (resize first last))
        end
      end
    and resize first last () =
      pending_resize := false;
      let curheight = int_of_string (cget wid CHeight) in
      if !newsize > curheight then begin
        if !debug then
         Log.f (sprintf "%s V %d %f %f newsize: %d"
                 (Widget.name wid) curheight first last !newsize);
        Text.configure wid [TextHeight !newsize]
      end
    in
    scroll, check
```

⟨*function* `Fit.bound_check` 359a⟩≡ (359b)

```
  let bound_check wid width =
    let stop_now = ref false in
    bind wid [[], Configure]
      (BindExtend([Ev_Width], (fun ei ->
        if !debug then
      Log.f (sprintf "Configure %s width is %d (max %d) (req %d)"
                   (Widget.name wid) ei.ev_Width width
                   (Winfo.reqwidth wid));
        if ei.ev_Width >= width then begin
      stop_now := true;
    bind wid [[], Configure] BindRemove
        end)));
    (fun () -> !stop_now)
```

⟨`display/fit.ml` 359b⟩≡

```
  open Printf
  open Tk


  ⟨constant Fit.debug 355g⟩

  ⟨function Fit.set_initial_width 355h⟩

  (* initial height: we have to grow at least to the number of
   * lines of text so that the entire text is visible. Moreover,
   * large lines will fold so adjust. Moreover, embedded window
   * provide height.
   *)
  ⟨function Fit.wheight 356a⟩

  ⟨function Fit.set_initial_height 356b⟩


  ⟨function Fit.fixed_horiz 356c⟩

  ⟨function Fit.horiz 357⟩
```

⟨*function* Fit.vert 358⟩

⟨*function* Fit.bound_check 359a⟩

F.8.6 display/fonts.mli

⟨*type* Fonts.fontInfo 360a⟩≡ (361a)
```
type fontInfo =
  | Family of string
  | Weight of string
  | Slant of string
  | FontIndex of int
  | FontDelta of int
```

⟨*type* Fonts.fontAttrs 360b⟩≡ (361a)
```
type fontAttrs = fontInfo list
```

⟨*type* Fonts.fontDesc 360c⟩≡ (361a)
```
type fontDesc = {
  pattern: Jpf_font.pattern;
  mutable pxlsz: int;
}
```

⟨*signature* Fonts.default 360d⟩≡ (361a)
```
val default: fontDesc ref
```

⟨*signature* Fonts.print_fontAttrs 360e⟩≡ (361a)
```
val print_fontAttrs: fontAttrs -> unit
```

⟨*signature* Fonts.merge 360f⟩≡ (361a)
```
val merge: fontDesc -> fontAttrs -> fontDesc
```

⟨*signature* Fonts.compute_tag 360g⟩≡ (361a)
```
val compute_tag: fontDesc -> string * Tk.options list
```

⟨*signature* Fonts.font_index 360h⟩≡ (361a)
```
val font_index: int -> int
```

⟨*signature* Fonts.pxlsz 360i⟩≡ (361a)
```
val pxlsz: int -> int
```

⟨*signature* Fonts.default_sizes 360j⟩≡ (361a)
```
(*-*)
val default_sizes: string list
```

⟨*signature* Fonts.reset 360k⟩≡ (361a)
```
val reset: unit -> unit
```

⟨display/fonts.mli 361a⟩≡

  ⟨*type* Fonts.fontInfo 360a⟩

  ⟨*type* Fonts.fontAttrs 360b⟩

  ⟨*type* Fonts.fontDesc 360c⟩

  ⟨*signature* Fonts.default 360d⟩

  ⟨*signature* Fonts.print_fontAttrs 360e⟩

  ⟨*signature* Fonts.merge 360f⟩
  ⟨*signature* Fonts.compute_tag 360g⟩

  ⟨*signature* Fonts.font_index 360h⟩
  ⟨*signature* Fonts.pxlsz 360i⟩

  ⟨*signature* Fonts.default_sizes 360j⟩

  ⟨*signature* Fonts.reset 360k⟩

F.8.7 display/fonts.ml

⟨type Fonts.fontDesc (./display/fonts.ml) 361b⟩≡ (364)

```
  (* Font manipulation *)

  type fontDesc =
      { pattern: Jpf_font.pattern;
        mutable pxlsz: int (* not pxlsz, but font index *)
      }
```

⟨type Fonts.fontInfo (./display/fonts.ml) 361c⟩≡ (364)

```
  type fontInfo =
     Family of string
   | Weight of string
   | Slant of string
   | FontIndex of int
   | FontDelta of int
```

⟨type Fonts.fontAttrs (./display/fonts.ml) 361d⟩≡ (364)

```
  type fontAttrs = fontInfo list
```

⟨*function* Fonts.copy 361e⟩≡ (364)

```
  let copy fd = { pattern= Jpf_font.copy fd.pattern;
          pxlsz= fd.pxlsz }
```

⟨*function* Fonts.print_fontAttrs 361f⟩≡ (364)

```
  let print_fontAttrs attrs =
    List.iter (function
      Family s -> prerr_string ("family: " ^ s ^ " ")
    | Weight s -> prerr_string ("weight: " ^ s ^ " ")
    | Slant s -> prerr_string ("slant: " ^ s ^ " ")
    | FontIndex i -> prerr_string ("index: " ^ string_of_int i ^ " ")
    | FontDelta i -> prerr_string ("delta: " ^ string_of_int i ^ " ")) attrs;
    prerr_endline "";
```

⟨*function* Fonts.merge 362a⟩≡ (364)
```
  (* Merge font attributes in a fontDesc *)
  let merge fd fil =
    let newfd = copy fd in
    List.iter (function
        Family s -> newfd.pattern.family <- Some s
      | Weight s -> newfd.pattern.weight <- Some s
      | Slant s -> newfd.pattern.slant <- Some s
      | FontIndex i -> newfd.pxlsz <- i
      | FontDelta n -> newfd.pxlsz <- newfd.pxlsz + n
      )
      fil;
    newfd
```

⟨*constant* Fonts.default_sizes 362b⟩≡ (364)
```
  (* List of authorized pixel sizes *)
  let default_sizes = ["8"; "10"; "12"; "14"; "15"; "16"; "18"; "20"; "24"; "26"; "28"]
```

⟨*constant* Fonts.sizes 362c⟩≡ (364)
```
  let sizes = ref (Array.of_list (List.map int_of_string default_sizes))
```

⟨*function* Fonts.get_index 362d⟩≡ (364)
```
  (* Given a size in pixels, find out the corresponding index array
     (which is the max of defined sized lower than argument)
   *)
  let get_index size =
    let len = Array.length !sizes in
    let rec walk n =
      if n >= len then len - 1
      else if !sizes.(n) > size then n-1
      else walk (succ n)
    in
    let idx = walk 0 in
    if idx < 0 then 0 else idx
```

⟨*constant* Fonts.base_index 362e⟩≡ (364)
```
  let base_index = ref (get_index 15)
```

⟨*function* Fonts.font_index 362f⟩≡ (364)
```
  (* Convert a pxlsz to an absolute font
   * (the base_index is always the absolute font 3 byte defintion of HTML)
   *)
  let font_index pxlsz =
    (get_index pxlsz) - (!base_index - 3)
```

⟨*function* Fonts.pxlsz 362g⟩≡ (364)
```
  (* Convert an absolute font to a pxlsz *)
  let pxlsz absfont =
    let font_idx = absfont + (!base_index - 3) in
    let safe_idx =
      if font_idx < 0 then 0
      else if font_idx >= Array.length !sizes then Array.length !sizes - 1
      else font_idx in
     !sizes.(safe_idx)
```

⟨*constant* `Fonts.tags` 363a⟩≡ (364)
```
(*
 * Tag names for fonts (this table is shared by all widgets)
 * We share tags for fonts, but this requires combinations of all
 * possible styles (weight, slant and size). The tag attribute is computed
 * on demand. Each widget must do its "tag configure" separately, since these
 * are not shared by all text widgets (even in a same class)
 *)


let tags = Hashtbl.create 37
```

⟨*constant* `Fonts.default` 363b⟩≡ (364)
```
let default = ref
    { pattern= Jpf_font.copy Jpf_font.empty_pattern;
      pxlsz = 3 }
```

⟨*function* `Fonts.compute_tag` 363c⟩≡ (364)
```
(* For a given fontDesc, return the name of the tags and its attributes *)
let rec compute_tag fd =
  let font_key =
    Jpf_font.string_of_pattern fd.pattern ^ string_of_int fd.pxlsz
  in
  try
    Hashtbl.find tags font_key
  with Not_found ->
    let tagdesc =
      let pxlsz = pxlsz fd.pxlsz in
      let pattern = {fd.pattern with pixelSize= Some pxlsz} in
      try
        let display =
          match Protocol.default_display () with
            "" -> None | x -> Some x
        in
        let fontid, fontname =
              (* find latin font *)
          let xlfd = Jpf_font.nearest_pixel_size display true pattern in
          let latin_f = Jpf_font.string_of_valid_xlfd xlfd in
          xlfd.family^xlfd.weight^xlfd.slant^(string_of_int pxlsz), latin_f
        in

        fontid, [Font fontname]
      with (* Invalid_argument f *) _ ->  (* font is not available *)
        Log.f (s_ "Font for %s is not available"
                 (Jpf_font.string_of_pattern pattern));
        if fd = !default
        then ("fixedfont", [Font "fixed"])
        else compute_tag !default
     in
     Hashtbl.add tags font_key tagdesc;
     tagdesc
```

⟨*function* `Fonts.reset` 363d⟩≡ (364)
```
(* Mapping with preferences : *fontPixels is also used to define our array
 * We use a mute preference handler to synchronize : this handler is called
 * after the loading of the resource file.
 *)
let reset () =
  Hashtbl.clear tags; (* since tags use font index *)
  let l = Tkresource.stringlist "fontPixels" default_sizes in
```

```
    sizes := Array.of_list (List.map (fun x ->
      try
        int_of_string x
      with
        e ->
      Log.f ("Fonts.reset error for size "^x);
      raise e) l);
    (* now we need to compute the base (we need to know some of Prefs internal)*)
    let b = Tkresource.string "prefDefaultFont" "" in
    if b = "" then base_index := get_index 15
    else
      let tokens = Mstring.split_str (fun c -> c='-') b in
      base_index := get_index (
         try int_of_string (List.nth tokens 6)
         with Failure "int_of_string" | Failure "nth" -> 15)
```

⟨display/fonts.ml 364⟩≡

```
  open I18n

  open Tk
  open Jpf_font
```

  ⟨*type Fonts.fontDesc (./display/fonts.ml)* 361b⟩

  ⟨*type Fonts.fontInfo (./display/fonts.ml)* 361c⟩

  ⟨*type Fonts.fontAttrs (./display/fonts.ml)* 361d⟩

  ⟨*function* `Fonts.copy` 361e⟩

  ⟨*function* `Fonts.print_fontAttrs` 361f⟩
  ;;

  ⟨*function* `Fonts.merge` 362a⟩

```
  (* HTML3.2 specifies that absolute font size are ranging from 1 to 7,
     the default basefont, used for "normal" text, being 3.

     The preference settings allow:
      - definition of list of pixel sizes
      - definition of default size
      - definition of header size.

     To compute the HTML fonts [1..7], we look for the default
     pixel size in the list of the given sizes: this defines the
     default base (3).

     We map these sizes to X Font Pxlsz, keeping some latitude for
     mapping the base. The lowest reasonable font is 8
   *)
```

  ⟨*constant* `Fonts.default_sizes` 362b⟩
  ⟨*constant* `Fonts.sizes` 362c⟩

  ⟨*function* `Fonts.get_index` 362d⟩

  ⟨*constant* `Fonts.base_index` 362e⟩

  ⟨*function* `Fonts.font_index` 362f⟩

```
  ⟨function Fonts.pxlsz 362g⟩

  ⟨constant Fonts.tags 363a⟩

  ⟨constant Fonts.default 363b⟩

  ⟨function Fonts.compute_tag 363c⟩

  ⟨function Fonts.reset 363d⟩
```

F.8.8 display/hr.mli

⟨*signature* Hr.create_named 365a⟩≡ (365b)

```
  val create_named:
    Widget.widget -> string -> Html.length -> int -> bool ->
    Widget.widget
```

⟨display/hr.mli 365b⟩≡

```
  ⟨signature Hr.create_named 365a⟩
```

F.8.9 display/hr.ml

⟨*function* Hr.create_named 365c⟩≡ (365d)

```
  (* When creating an HR in a nested window (eg table cell), reqwidth is
     probably the width of 1 character
   *)
  let create_named top name length height solid =
    let fr = Frame.create_named top name [] in
    let width = match length with
      Nolength | LengthRel _-> truncate (float (Winfo.reqwidth top) *. 0.95)
    | LengthRatio r -> truncate (float (Winfo.reqwidth top) *. r)
    | LengthPixels n -> n
    in
    Frame.configure fr [Width (Pixels width)];
    if solid then
      Frame.configure fr [BorderWidth (Pixels 0); Height (Pixels height)]
    else
      Frame.configure fr [Relief Groove;
              BorderWidth (Pixels 2); Height (Pixels (height+2))];
    fr
```

⟨display/hr.ml 365d⟩≡

```
  open Tk
  open Html

  ⟨function Hr.create_named 365c⟩
```

F.8.10 display/htbind.mli

⟨display/htbind.mli 365e⟩≡

```

  class virtual active :
    unit ->
    object
      method virtual binder :
```

```
      (Tk.modifier list * Tk.xEvent) list -> Tk.bindAction -> unit
    method virtual getlink : Tk.eventInfo -> Hyper.link
    method virtual highlight : bool -> unit
    method init : Viewers.context -> unit
    method virtual markused : Tk.eventInfo -> unit
    method virtual widget : Widget.widget
  end

class virtual hypertext :
  Widget.widget ->
  object
    method binder :
      (Tk.modifier list * Tk.xEvent) list -> Tk.bindAction -> unit
    method virtual getlink : Tk.eventInfo -> Hyper.link
    method virtual getrange : Tk.index -> Tk.index * Tk.index
    method highlight : bool -> unit
    method init : Viewers.context -> unit
    method markused : Tk.eventInfo -> unit
    method widget : Widget.widget
  end

class directmap :
  Widget.widget * Hyper.link ->
  object
    val link : Hyper.link
    method binder :
      (Tk.modifier list * Tk.xEvent) list -> Tk.bindAction -> unit
    method getlink : Tk.eventInfo -> Hyper.link
    method highlight : bool -> unit
    method init : Viewers.context -> unit
    method markused : Tk.eventInfo -> unit
    method widget : Widget.widget
  end

class servermap :
  Widget.widget * Hyper.link ->
  object
    val link : Hyper.link
    method binder :
      (Tk.modifier list * Tk.xEvent) list -> Tk.bindAction -> unit
    method getlink : Tk.eventInfo -> Hyper.link
    method highlight : bool -> unit
    method init : Viewers.context -> unit
    method markused : Tk.eventInfo -> unit
    method widget : Widget.widget
  end

class formmap :
  Widget.widget * (int * int -> Hyper.link) ->
  object
    val formlink : int * int -> Hyper.link
    method binder :
      (Tk.modifier list * Tk.xEvent) list -> Tk.bindAction -> unit
    method getlink : Tk.eventInfo -> Hyper.link
    method highlight : bool -> unit
    method init : Viewers.context -> unit
    method markused : Tk.eventInfo -> unit
    method widget : Widget.widget
  end
```

F.8.11 display/htbind.ml

⟨display/htbind.ml 367⟩≡

```
  (* Bindings for hypernavigation *)
  open Printf
  open Tk
  open Hyper
  open Viewers

  (*
  An active object is assumed to have the following methods:
  *)


  class  virtual active () =
   object (self)
   method virtual widget : Widget.widget
        (* returns the widget to which an hypermenu can be attached *)
   method virtual getlink : eventInfo -> Hyper.link
        (* returns the link pointed to by the object *)
   method virtual binder : (modifier list * xEvent) list -> bindAction -> unit
        (* binds events on the object *)
   method virtual highlight : bool -> unit
        (* user feedback (mostly cursor) indicating that object is active *)
   method virtual markused : eventInfo -> unit
        (* say that we've activated this link *)

   method init (ctx : Viewers.context) =
    (* Install all navigation bindings *)
    List.iter (fun hyname ->
      try
        let _hyperfTODO = List.assoc hyname ctx#hyper_funs in
        self#binder (Glevents.get hyname)
      (BindSet ([Ev_MouseX; Ev_MouseY],
            (fun ei ->
               try let link = self#getlink ei in
                    self#markused ei;
                    ctx#invoke hyname link;
               with Not_found -> ())))
      with
        Not_found -> ())
      ["goto"; "save"; "gotonew"];

    (* Install the menu (created by need only) *)
    let menulink = ref None
    and menuei = ref None in
    let hypermenu = Frx_misc.autodef (fun () ->
       let m = Menu.create_named self#widget "hypermenu" [] in
          (* The first entry has its text replaced by the link url *)
          Menu.add_command m [Label ""];
          Menu.add_separator m;
      List.iter (fun (fname, f) ->
               if f.hyper_visible then
                Menu.add_command m
                  [Label f.hyper_title;
                   Command (fun () ->
                     match !menuei, !menulink with
                   Some ei, Some link ->
                     self#markused ei;
                     ctx#invoke fname link
                  | _, _ -> ())])
          ctx#hyper_funs;
    m) in
  self#binder (Glevents.get "hypermenu")
   (BindSet ([Ev_MouseX; Ev_MouseY; Ev_RootX; Ev_RootY],
       (fun ei ->
     try
       let link = self#getlink ei in
       menuei := Some ei; menulink := Some link;
       let m = hypermenu() in
       Menu.configure_command m (Number 1) [Label (Hyper.string_of link)];
       Menu.popup m ei.ev_RootX ei.ev_RootY
     with
       Not_found -> ())));

  (* Install the pointsto internal bindings *)
  self#binder [[], Enter]
    (BindExtend ([Ev_MouseX; Ev_MouseY],
         (fun ei ->
           try
             let link = self#getlink ei in
             self#highlight true;
             ctx#invoke "pointsto" link
           with Not_found -> ())));
  let fakehlink = Hyper.default_link "" in
  self#binder [[], Leave]
    (BindSet ([Ev_MouseX; Ev_MouseY],
          (fun _ei ->
        self#highlight false;
        ctx#invoke "clearpointsto" fakehlink)))
end

(*
 * The various active objects
 *)

(* Text widget with anchors marked as tags *)

class  virtual hypertext (thtml) =
 object (self)
  inherit active () as super
  (* val thtml = thtml *)  (* keep our own copy *)

  method widget = thtml

  method virtual getlink : eventInfo -> Hyper.link

  method binder = Text.tag_bind thtml "anchor"

  method highlight flag =
    if flag then
      Text.configure thtml [Cursor (XCursor "hand2")]
    else
      Text.configure thtml [Cursor (XCursor "xterm")]

  method virtual getrange : index -> index * index

  method markused ei =
     (* The index of the click position *)
     let i =
       Text.index thtml (TextIndex (AtXY (ei.ev_MouseX,ei.ev_MouseY), [])) in
     (* Tags at this place *)
     let s,e = self#getrange i in
       Text.tag_add thtml "visited" (TextIndex (s,[])) (TextIndex (e,[]))

  (* we don't get Enter/Leave when tags are contiguous, so the
     pointed link displayed in pointsto is no always correct
     Thus, extend initialisation to bind pointsto on motion
   *)
  method! init ctx =
    super#init ctx;
    self#binder [[], Motion]
      (BindSet ([Ev_MouseX; Ev_MouseY],
        (fun ei ->
          try
            let link = self#getlink ei in
            ctx#invoke "pointsto" link
          with
            Not_found -> ())))
end


(* embedded objects with direct map *)
class directmap (frame, link) =
 object (_self)
  inherit active ()
  (* val frame = frame *)
  method widget = frame
  val link = (link : Hyper.link)
  method getlink (_ei : eventInfo) = link
  method binder = bind frame
  method highlight (_flag : bool) = ()  (* we already set up the cursor *)
  method markused _ei =
    Frame.configure frame [Relief Sunken]
end

(* embedded objects with server map (ISMAP) *)
(* pointsto will get some arbitrary value for x,y... *)
class servermap (frame,link) =
 object (_self)
  inherit active ()
  inherit! directmap (frame, link)
  method! getlink ei =
    {h_uri = sprintf "%s?%d,%d" link.h_uri
                                ei.ev_MouseX ei.ev_MouseY;
     h_context = link.h_context;
     h_method = GET;
     h_params = link.h_params}
end

(* embedded objects with form submission *)
class formmap (frame,formlink) =
 object (_self)
  inherit active ()
  (* val frame = frame *)
  method widget = frame
  val formlink = (formlink : int * int -> Hyper.link)
  method getlink ei = formlink (ei.ev_MouseX, ei.ev_MouseY)
  method binder = bind frame
  method highlight (_flag : bool) = ()  (* we already set up the cursor *)
```

```
    method markused _ei =
      Frame.configure frame [Relief Sunken]
  end


  (* Client side image maps are defined in Cmap *)
```

F.8.12 display/source.mli

⟨*signature* Source.annotate 370a⟩≡ (370b)
```
  val annotate:
    Widget.widget -> (Tk.textTag * Html.location) list -> unit
```

⟨display/source.mli 370b⟩≡

  ⟨*signature* Source.annotate 370a⟩

  ⟨*signature* Source.view 192f⟩

F.8.13 display/source.ml

⟨*function* Source.annotate 370c⟩≡ (370d)
```
  (* HTML source viewer/editor *)
  let annotate txt =
    Hashtbl.iter (fun elem _  ->
      let color = Tkresource.string (sprintf "Source<%s>" elem) "white" in
      Text.tag_configure txt elem [Background (NamedColor color)])
      !Dtd.current.Dtd.contents;
      (fun annotations ->
        List.iter (function (name,Loc(s,e)) ->
      let idxs = abs_index s
      and idxe = abs_index e in
      Text.tag_add txt name idxs idxe)
        annotations)
```

⟨display/source.ml 370d⟩≡
```

  open Fpath_.Operators

  open I18n
  open Printf
  open Tk
  open Frx_text
  open Document
  open Html
```

  ⟨*function* Source.annotate 370c⟩

  ⟨*function* Source.view 192g⟩

F.8.14 display/ctext.mli

⟨*signature* Ctext.create 370e⟩≡ (371b)
```
  (* [create parent opts nav_keys] creates a text widget
   * with "pixel scrolling". Based on a trick learned from Steve Ball.
   * Returns (frame widget, text widget).
   *)
  val create :
    Widget.widget -> Tk.options list -> bool ->
    Widget.widget * Widget.widget
```

⟨*signature* `Ctext.init` 371a⟩≡ (371b)
```
  val init : unit -> unit
```

⟨`display/ctext.mli` 371b⟩≡

  ⟨*signature* `Ctext.create` 370e⟩

  ⟨*signature* `Ctext.init` 371a⟩

F.8.15 `display/ctext.ml`

⟨*constant* `Ctext.tag_name` 371c⟩≡ (374a)
```
  let tag_name = "CTEXT_RO"
```

⟨*function* `Ctext.navigation_keys` 371d⟩≡ (374a)
```
  let navigation_keys tx =
    let tags = bindtags_get tx in
    match tags with
    | (WidgetBindings t)::l when t = tx ->
        Canvas.configure (Winfo.parent t) [YScrollIncrement (Pixels 15)];
        bindtags tx ((WidgetBindings tx) :: (TagBindings tag_name) :: l)
    | _ -> ()
```

⟨*function* `Ctext.create` 371e⟩≡ (374a)
```
  let create top opts navigation =
    let f = Frame.create_named top "smoothf" [] in
    let lf = Frame.create_named f "left" [] in
    let rf = Frame.create_named f "right" [] in
    let c = Canvas.create_named lf "smoothc" [BorderWidth (Pixels 0); TakeFocus true]
    and xscroll = Scrollbar.create_named lf "x" [Orient Horizontal]
    and yscroll = Scrollbar.create_named rf "y" [Orient Vertical]
    and secret = Frame.create_named rf "secret" []
    in
    (* automatic scrollbars *)
    let has_x = ref false
    and has_y = ref false
    in
    let putx () =
      pack [xscroll] [Before c; Side Side_Bottom; Fill Fill_X];
      pack [secret] [Before yscroll; Side Side_Bottom];
      has_x := true
    and remx () =
      Pack.forget [xscroll; secret];
      has_x := false
    and puty () =
      pack [rf] [Before lf; Side Side_Right; Fill Fill_Y];
      has_y := true
    and remy () = Pack.forget [rf]; has_y := false
    in

    let wrap_scroll isthere put rem scrollcmd   =
      fun first last ->
        scrollcmd first last;
        if !isthere then
         if first = 0.0 && last = 1.0 then rem() else ()
        else
         if first <> 0.0 || last <> 1.0 then put() else ()
    in
    let t = Text.create_named c "smootht" (BorderWidth(Pixels 0) :: opts) in
      if navigation then navigation_keys t;

      (* Make the text widget an embedded canvas object *)
      ignore (
        Canvas.create_window c (Pixels 0) (Pixels 0)
         [Anchor NW; Window t; Tags [Tag "main"]]);
      Canvas.focus c (Tag "main");
      Canvas.configure c
      [YScrollCommand (wrap_scroll has_y puty remy (Scrollbar.set yscroll))];
      (* The horizontal scrollbar is directly attached to the
       * text widget, because h scrolling works properly *)
      Scrollbar.configure xscroll [ScrollCommand (Text.xview t)];
      (* But vertical scroll is attached to the canvas *)
      Scrollbar.configure yscroll [ScrollCommand (Canvas.yview c)];
      let scroll, _check = Fit.vert t in
      Text.configure t [
         XScrollCommand (wrap_scroll has_x putx remx (Scrollbar.set xscroll));
          YScrollCommand (fun first last ->
            scroll first last;
         let x,y,w,h = Canvas.bbox c [Tag "main"] in
           Canvas.configure c
                [ScrollRegion (Pixels x, Pixels y, Pixels w, Pixels h)]);
          ];
      (* B2 Scrolling : based on std script text.tcl
       * Since t has the focus, it will handle the event, even if we play
       * with bindtags. Thus ev_MouseX/Y are given in the "full" text.
       * Moreover, as the text scrolls, the re-positionning of the text
       * affects the event fields (non-monotonicity !)
       * However, since "scan" is interested only in relative positions,
       * we can use root coordinates directly
       *)
      let x = ref 0
      and y = ref 0
      in
      bind t [[], ButtonPressDetail 2]
        (BindSetBreakable ([Ev_RootX; Ev_RootY],
          (fun ei ->
            x := ei.ev_RootX;
            y := ei.ev_RootY;
            Canvas.scan_mark c !x !y)));
      bind t [[Button2], Motion]
        (BindSetBreakable ([Ev_RootX; Ev_RootY],
          (fun ei ->
            let dx = ei.ev_RootX
            and dy = ei.ev_RootY
            in
            if dx <> !x || dy <> !y then
              Canvas.scan_dragto c dx dy)));

      bind c [[],Configure] (BindSet ([Ev_Width], (fun ei ->
        Canvas.configure_window c (Tag "main") [Width (Pixels ei.ev_Width)])));

      pack [c] [Side Side_Left; Fill Fill_Both; Expand true];
      pack [lf] [Side Side_Left; Fill Fill_Both; Expand true];
      (* pack [secret] [Side Side_Bottom]; *)
      pack [yscroll] [Side Side_Top; Fill Fill_Y; Expand true];
```

```
    (* pack [rf] [Side Side_Right; Fill Fill_Y]; *)
    (* pack [xscroll] [Side Side_Bottom; Fill Fill_X]; *)
    f, t
```

⟨*function* `Ctext.init` 373⟩≡ (374a)

```
  (* We use Mod1 instead of Meta or Alt *)
  let init () =
    List.iter (function ev ->
             tag_bind tag_name ev
                 (BindSetBreakable ([Ev_Widget],
                               (fun ei -> page_up ei.ev_Widget; break()))))
        [
         [[], KeyPressDetail "BackSpace"];
         [[], KeyPressDetail "Delete"];
         [[], KeyPressDetail "Prior"];
         [[], KeyPressDetail "b"];
         [[Mod1], KeyPressDetail "v"]
        ];
    List.iter (function ev ->
             tag_bind tag_name ev
                 (BindSetBreakable ([Ev_Widget],
                               (fun ei -> page_down ei.ev_Widget; break()))))
        [
         [[], KeyPressDetail "space"];
         [[], KeyPressDetail "Next"];
         [[Control], KeyPressDetail "v"]
        ];
    List.iter (function ev ->
             tag_bind tag_name ev
                 (BindSetBreakable ([Ev_Widget],
                               (fun ei -> line_up ei.ev_Widget; break()))))
        [
         [[], KeyPressDetail "Up"];
         [[Mod1], KeyPressDetail "z"]
        ];
    List.iter (function ev ->
             tag_bind tag_name ev
                 (BindSetBreakable ([Ev_Widget],
                               (fun ei -> line_down ei.ev_Widget; break()))))
        [
         [[], KeyPressDetail "Down"];
         [[Control], KeyPressDetail "z"]
        ];

    List.iter (function ev ->
             tag_bind tag_name ev
                 (BindSetBreakable ([Ev_Widget],
                               (fun ei -> top ei.ev_Widget; break()))))
        [
         [[], KeyPressDetail "Home"];
         [[Mod1], KeyPressDetail "less"]
        ];

    List.iter (function ev ->
             tag_bind tag_name ev
                 (BindSetBreakable ([Ev_Widget],
                               (fun ei -> bottom ei.ev_Widget; break()))))
        [
         [[], KeyPressDetail "End"];
         [[Mod1], KeyPressDetail "greater"]
        ]
```

⟨display/ctext.ml 374a⟩≡

```
  (* A trick by Steve Ball to do pixel scrolling on text widgets *)
  (* USES frx_fit *)
  open Tk

  ⟨constant Ctext.tag_name 371c⟩

  ⟨function Ctext.navigation_keys 371d⟩


  ⟨function Ctext.create 371e⟩


  (* Same as frx_text, but work on canvas instead of text for scrolling *)
  let page_up tx   =  Canvas.yview (Winfo.parent tx) (ScrollPage (-1))
  and page_down tx =  Canvas.yview (Winfo.parent tx) (ScrollPage 1)
  and line_up tx   =  Canvas.yview (Winfo.parent tx) (ScrollUnit (-1))
  and line_down tx =  Canvas.yview (Winfo.parent tx) (ScrollUnit 1)
  and top tx = Canvas.yview (Winfo.parent tx) (MoveTo 0.0)
  and bottom tx = Canvas.yview (Winfo.parent tx) (MoveTo 1.0)

  ⟨function Ctext.init 373⟩
```

F.8.16 display/htmlfmt.ml

⟨Htmlfmt.gattr *color cases* 374b⟩+≡ (26c) ◁157b

```
  | BgColor of string
```

⟨display/htmlfmt.ml 374c⟩≡

```
  (* HTML  "display device" *)

  ⟨type Htmlfmt.gattr 26c⟩

  ⟨type Htmlfmt.formatterSpec 129d⟩

  ⟨type Htmlfmt.formatter 26d⟩


  ⟨type Htmlfmt.input_kind 173a⟩

  class  virtual form_behaviour () = object
      method virtual add_get : input_kind -> (unit -> (string * string) list) -> unit
      method virtual add_reset : (unit -> unit) -> unit
      method virtual submit : (string * string) list -> Hyper.link
      method virtual single_submit : Hyper.link option
      method virtual reset : unit
  end

  ⟨type Htmlfmt.width_constraint 152b⟩

  (*
  module type TableDisplay = sig
      type cell_type = HeaderCell | DataCell
      type t = {
        table_master : Widget.widget;
        add_col : Html.tag -> unit;
        open_row : Html.tag -> unit;
      close_row : unit -> unit;
      close_table : unit -> unit;
      new_cell :
      cell_type -> Html.tag -> Widget.widget -> string -> width_constraint;
      bound : unit -> bool
      }

    val create : Widget.widget -> Html.tag -> width_constraint -> t

    val topwidth : Widget.widget -> int
 end
*)

(*
module type FormDisplay = sig
   (* A form manager *)
   type t = {
     text_input : Widget.widget -> Html.tag -> unit;
     (* [text_input top tag] *)
     checkbox_input : Widget.widget ->  Html.tag -> unit;
     (* [input top tag] *)
     radio_input : Widget.widget ->  Html.tag -> unit;
     (* [input top tag] *)
     image_input : Widget.widget ->  Html.tag -> Embed.embobject;
     (* [input top tag] *)
     submit_input : Widget.widget ->  Html.tag -> unit;
     (* [input top tag] *)
     reset_input : Widget.widget ->  Html.tag -> unit;
     (* [input top tag] *)
     select : Widget.widget -> (string * string * bool) list -> Html.tag -> unit;
     (* [select top elements tag] *)
     textarea:  Widget.widget -> string -> Html.tag -> unit
     (* [textarea top initial attrs] *)
     }

   val create : string -> form_behaviour -> Viewers.context -> t
      (* [create base behaviour ctx] *)

 end
*)

(*
module type GfxHTML = sig
 val create :
    (unit -> string) ->
    formatterSpec -> Widget.widget -> Viewers.context ->
      formatter * Widget.widget
 end
*)
```

F.8.17 display/imgload.mli

⟨*type* Imgload.mode 375⟩≡ (376c)

```
  type mode =
    | DuringDoc
    | AfterDocAuto
    | AfterDocManual
```

```
⟨signature Imgload.mode 376a⟩≡                                         (376c)
  val mode : mode ref

⟨signature Imgload.no_images 376b⟩≡                                    (376c)
  val no_images : bool ref

⟨display/imgload.mli 376c⟩≡

  ⟨type Imgload.mode 375⟩

  ⟨signature Imgload.mode 376a⟩
  ⟨signature Imgload.no_images 376b⟩
  ⟨signature Imgload.gif_anim_auto 218e⟩

  ⟨class Imgload.loader signature 172d⟩

  ⟨signature Imgload.create 172c⟩
```

F.8.18 display/imgload.ml

```
⟨type Imgload.mode (./display/imgload.ml) 376d⟩≡                       (380b)
  (* Images are embedded objects, with a twist *)
  type mode = DuringDoc | AfterDocAuto | AfterDocManual

⟨constant Imgload.mode 376e⟩≡                                          (380b)
  (* Preference settings *)
  let mode = ref AfterDocManual

⟨constant Imgload.no_images 376f⟩≡                                     (380b)
  let no_images = ref false

⟨function Imgload.display 376g⟩≡                                       (380b)
  (* Utilities *)
  let display (caps : < Cap.network ; .. >) (emb : Embed.obj)
      (i : Tkanim.imageType) =
    let prop = ref false in
    begin
      if Winfo.exists emb.embed_frame then
        match emb.embed_map with
        (* kill'em all *)
        | ClientSide hlink -> begin
            try
              List.iter Tk.destroy (Winfo.children emb.embed_frame);
              let w, h =
                match i with
                | Still x -> begin
                    match x with
                    | ImageBitmap i -> (Imagebitmap.width i, Imagebitmap.height i)
                    | ImagePhoto i -> (Imagephoto.width i, Imagephoto.height i)
                    | _ -> failwith "invalid image"
                  end
                | Animated anm -> (Tkanim.width anm, Tkanim.height anm)
              in
              let c =
                Canvas.create emb.embed_frame
                  [ Width (Pixels w); Height (Pixels h) ]
              in
              prop := true;
              (*fit the size to the image*)
              Tk.bindtags c (WidgetBindings emb.embed_frame :: Tk.bindtags_get c);
              Tk.pack [ c ] [];
              let ii =
                Canvas.create_image c (Pixels 0) (Pixels 0) [ Anchor NW ]
              in
              begin
                match i with
                | Still x -> Canvas.configure_image c ii [ x ]
                | Animated anm -> begin
                    let f = Tkanim.animate_canvas_item c ii anm in
                    (* binding on c is bad... *)
                    Tk.bind c (Glevents.get "stopanim")
                      (BindSet ([], fun _ -> f false));
                    Tk.bind c
                      (Glevents.get "restartanim")
                      (BindSet ([], fun _ -> f true));
                    (* I am sure it doesn't work *)
                    Canvas.configure c [ Cursor (XCursor "watch") ];
                    if !gif_anim_auto then f false
                  end
              end;
              (* now we have the image displayed in a canvas.
                 and we can create the client side map *)
              let uri = Hyper.resolve hlink in
              let name =
                match uri.uri_fragment with
                | None -> uri.uri_url
                | Some frag -> Printf.sprintf "%s#%s" uri.uri_url frag
              in
              match Maps.get name with
              | KnownMap m -> Cmap.gfx_mode emb m c
              | RequestedMap event ->
                  Frx_synth.bind c event (fun c ->
                      match Maps.get name with
                      | RequestedMap _ ->
                          Log.f "INTERNAL ERROR: delayed_client_side"
                      | KnownMap m -> Cmap.gfx_mode emb m c)
            with
            | e ->
                Logs.err (fun m ->
                    m "INTERNAL ERROR in display (%s)" (Printexc.to_string e))
          end
        | _ ->
            (* in all other cases, get the alt label, and configure it *)
            (* WARNING: there may be an hypermenu here *)
            Winfo.children emb.embed_frame
            |> List.iter (fun w ->
                   match Winfo.class_name w with
                   | "Label" ->
                       (* remove its border *)
                       Label.configure w [ BorderWidth (Pixels 0) ];
                       begin
                         match i with
                         | Still x ->
                             Label.configure w [ x ];
                             begin
                               match x with
                               | ImageBitmap _
                               | ImagePhoto _ ->
                                   prop := true
```

```
                              (*fit the size to the image*)
                              | _ -> ()
                              (*We cannot restore the origianl size of the window...*)
                            end;
                            (* Utility to copy the img url in the selection buffer *)
                            Tk.bind w
                              (Glevents.get "copyimgurl")
                              (BindSet
                                 ( [],
                                   fun _ ->
                                     emb.embed_context#invoke "copy"
                                       emb.embed_hlink ));
                            (* Updating an image *)
                            begin
                              try
                                let url =
                                  Lexurl.make
                                    (Hyper.resolve emb.embed_hlink).uri_url
                                in
                                Tk.bind w
                                  (Glevents.get "updateimage")
                                  (BindSet ([], fun _ -> Img.update caps url))
                              with
                              | Url.Url_Lexing _ -> ()
                            end
                        | Animated anm -> begin
                            let f = Tkanim.animate w anm in
                            Tk.bind w (Glevents.get "stopanim")
                              (BindSet ([], fun _ -> f false));
                            Tk.bind w
                              (Glevents.get "restartanim")
                              (BindSet ([], fun _ -> f true));
                            Label.configure w [ Cursor (XCursor "watch") ];
                            if !gif_anim_auto then f false;
                            prop := true (*fit the size to the image*)
                          end
                      end
                | "Canvas" -> Tk.destroy w (* delete the progress meter *)
                | _ -> ())
    end;
    if !prop then Pack.propagate_set emb.embed_frame true
```

⟨*function* Imgload.put_alt 378⟩≡ (380b)

```
  (* put up the alternate text *)
  let put_alt (emb : Embed.obj) =
    let m = Label.create_named emb.embed_frame "alt" [ Text emb.embed_alt ] in
    (* make sure all bindings we put on the frame are attached there *)
    Tk.bindtags m (WidgetBindings emb.embed_frame :: Tk.bindtags_get m);
    Tk.pack [ m ] [ Fill Fill_Both; Expand true ];
    if not (Pack.propagate_get emb.embed_frame) then begin
      (* with width and height *)
      if Winfo.reqwidth emb.embed_frame < Winfo.reqwidth m then
        Frame.configure emb.embed_frame [ Width (Pixels (Winfo.reqwidth m)) ];
      if Winfo.reqheight emb.embed_frame < Winfo.reqheight m then
        Frame.configure emb.embed_frame [ Height (Pixels (Winfo.reqheight m)) ]
        (* Buggy
            let wf = Winfo.reqwidth emb.embed_frame
            and wm = Winfo.reqwidth m
            and hf = Winfo.reqheight emb.embed_frame
            and hm = Winfo.reqheight m
```

```
          in
          if wf < wm || hf < hm then begin
            bind emb.embed_frame [[], Enter] (BindExtend ([], (fun _ ->
          if wf < wm then
            Frame.configure emb.embed_frame [Width (Pixels wm)];
          if hf < hm then
            Frame.configure emb.embed_frame [Height (Pixels hm)])));
            bind emb.embed_frame [[], Leave] (BindExtend ([], (fun _ ->
          Frame.configure emb.embed_frame [Width (Pixels wf)];
          Frame.configure emb.embed_frame [Height (Pixels hf)])))
          end
      *)
    end
```

⟨*function* Imgload.make_auto 379a⟩≡ (380b)

```
  (* for delayed load, add binding *)
  let make_auto (caps : < Cap.network >) delayed (emb : Embed.obj) =
    try
      let url = (Www.make emb.embed_hlink).www_url in
      Tk.bind emb.embed_frame (Glevents.get "loadimage")
        (BindSet ([], fun _ -> Img.ImageScheduler.flush_one caps delayed url))
    with
    | e ->
        Logs.warn (fun m ->
            m "Can't compute image link (%s)" (Printexc.to_string e))
```

⟨*function* Imgload.make_map 379b⟩≡ (380b)

```
  (* If the object is clickable, make it visible *)

  let make_map (emb : Embed.obj) =
    let visible =
      [
        Tk.BorderWidth (Pixels (Tkresource.int "clickableBorderWidth" 2));
        Relief (Tkresource.relief "clickableRelief" Raised);
        Cursor (XCursor (Tkresource.string "clickableCursor" "hand2"));
        Background (NamedColor (Tkresource.string "clickableBackground" "white"));
      ]
    and visible_map =
      [
        Tk.BorderWidth (Pixels (Tkresource.int "clickableBorderWidth" 2));
        Relief (Tkresource.relief "clickableRelief" Raised);
        Cursor (XCursor (Tkresource.string "clickableMapCursor" "left_ptr"));
        Background (NamedColor (Tkresource.string "clickableBackground" "white"));
      ]
    in
    let reconfigure_frame f =
      let internal_width =
        try int_of_string (Tk.cget f CWidth) with
        | _ -> 0
      and internal_height =
        try int_of_string (Tk.cget f CHeight) with
        | _ -> 0
      and border_width =
        try int_of_string (Tk.cget f CBorderWidth) with
        | _ -> 0
      in
      if internal_width = 0 || internal_height = 0 then ()
      else
        Frame.configure f
          [
          Width (Pixels (internal_width + (border_width * 2)));
          Height (Pixels (internal_height + (border_width * 2)));
        ]
    in
    match emb.embed_map with
    | ClientSide hlink ->
       Frame.configure emb.embed_frame visible_map;
       reconfigure_frame emb.embed_frame;
       (* At this moment, we assume that we are in alt mode.
          If the image gets loaded, the label gets destroyed and
          the callback will never be invoked. Instead, it will
          be called from "display" *)
       begin
         try
          match Winfo.children emb.embed_frame with
          | [ l ] when Winfo.class_name l = "Label" ->
              let uri = Hyper.resolve hlink in
              let name =
                match uri.uri_fragment with
                | None -> uri.uri_url
                | Some frag -> Printf.sprintf "%s#%s" uri.uri_url frag
              in
              begin
                match Maps.get name with
                | KnownMap m -> Cmap.alt_mode emb m l
                | RequestedMap event ->
                    Frx_synth.bind l event (fun l ->
                        match Maps.get name with
                        | RequestedMap _ ->
                            Log.f "INTERNAL ERROR: delayed_client_side"
                        | KnownMap m -> Cmap.alt_mode emb m l)
              end
          | _ -> Log.f "make_map. children not a label"
        with
        | _ -> ()
       end
    | ServerSide link ->
       Frame.configure emb.embed_frame visible;
       reconfigure_frame emb.embed_frame;
       (new Htbind.servermap (emb.embed_frame, link))#init emb.embed_context
    | Direct link ->
       Frame.configure emb.embed_frame visible;
       reconfigure_frame emb.embed_frame;
       (new Htbind.directmap (emb.embed_frame, link))#init emb.embed_context
    | NoMap -> ()
    | FormMap getlink ->
       Frame.configure emb.embed_frame visible;
       reconfigure_frame emb.embed_frame;
       (new Htbind.formmap (emb.embed_frame, getlink))#init emb.embed_context
```

⟨*function* Imgload.create 380a⟩≡ (380b)

```
  let create () : loader =
    if !no_images then new loader ()
    else
      match !mode with
      | DuringDoc -> (new synchronous () :> loader)
      | AfterDocAuto -> (new auto () :> loader)
      | AfterDocManual -> (new manual () :> loader)
```

⟨display/imgload.ml 380b⟩≡

```
⟨type Imgload.mode (./display/imgload.ml) 376d⟩

⟨constant Imgload.mode 376e⟩
⟨constant Imgload.no_images 376f⟩
⟨constant Imgload.gif_anim_auto 218f⟩

⟨function Imgload.display 376g⟩

⟨function Imgload.put_alt 378⟩

⟨function Imgload.make_auto 379a⟩

(* for manual load, add binding
   let make_manual emb =
     try
       let url = (Www.make emb.embed_hlink).www_url in
       bind emb.embed_frame
         (Glevents.get "loadimage")
         (BindSet ([], (fun _ -> activate emb)))
     with
       e -> Log.f (sprintf "Can't compute image link (%s)" (Printexc.to_string e))
*)

⟨function Imgload.make_map 379b⟩

(* The default behavior, for no_images *)
class loader () =
  object (self)
    val mutable loaded = Www.UrlSet.empty

    (* default no_image implem *)
    method add_image (_caps : < Cap.network >) (emb : Embed.obj) : unit =
      put_alt emb;
      (* make the alt widget*)
      make_map emb (* and possible bindings *)

    (* flush when document is loaded *)
    method flush_images = ()

    (* manual flush *)
    method load_images = ()

    method private add_loaded (url : Url.t) : unit =
      loaded <- Www.UrlSet.add url loaded

    method private activate (caps : < Cap.network >) (emb : Embed.obj) =
      Logs.debug (fun m -> m "Activating image");
      try
        Img.get caps emb.embed_context#base emb.embed_hlink
          (fun url i ->
            display caps emb i;
            self#add_loaded url)
          (Tk_progress.meter emb.embed_frame)
      with
      | e ->
          Logs.warn (fun m -> m "Can't load image (%s)" (Printexc.to_string e))

    (* called when ?? *)
    method update_images (caps : < Cap.network >) : unit =
      Www.UrlSet.iter (Img.update caps) loaded
  end

(* for DuringDoc *)
class synchronous () =
  object
    inherit loader () as super

    method! add_image (caps : < Cap.network >)(emb : Embed.obj) =
      super#add_image caps emb;
      super#activate caps emb
  end

(* for AfterDocAuto *)
class auto () =
  object (self)
    inherit loader () as super
    val q = Img.ImageScheduler.new_delayed ()

    method! add_image (caps : < Cap.network>) (emb : Embed.obj) =
      super#add_image caps emb;
      try
        let wr = Www.make emb.embed_hlink in
        wr.www_headers <- "Accept: image/*" :: wr.www_headers;
        Img.ImageScheduler.add_delayed q wr emb.embed_context#base
          (fun url i ->
            display caps emb i;
            self#add_loaded url)
          (Tk_progress.meter emb.embed_frame)
      with
      | e ->
          Logs.warn (fun m ->
              m "Can't compute image link (%s)" (Printexc.to_string e))

    method! flush_images = Img.ImageScheduler.flush_delayed q
  end

(* for AfterDocManual *)
class manual () =
  object
    inherit auto () as super

    method! add_image (caps : < Cap.network >) (emb : Embed.obj) =
      super#add_image caps emb;
      make_auto caps q emb

    method! flush_images = ()
    method! load_images = Img.ImageScheduler.flush_delayed q
  end

⟨function Imgload.create 380a⟩
```

F.8.19 display/html_form.mli

⟨display/html_form.mli 382⟩≡

```
  class behaviour :
    string * Html.tag * string option * (string -> string) ->
    object
    val action : string
    val mutable elem_reset : (unit -> unit) list
    val mutable elem_value :
      (Htmlfmt.input_kind * (unit -> (string * string) list)) list
    val encoding : string
    val mutable entries : int
    val fmethod : string
    val h_params : (string * string) list
    method add_get :
      Htmlfmt.input_kind -> (unit -> (string * string) list) -> unit
    method add_reset : (unit -> unit) -> unit
    method reset : unit
    method single_submit : Hyper.link option
    method submit : (string * string) list -> Hyper.link
  end

(*
module Make :
  functor (FormDisplay : Htmlfmt.FormDisplay) ->
    sig
      val init :
        < add_tag : string ->
                    (Htmlfmt.formatter -> Html.tag -> unit) ->
                    (Htmlfmt.formatter -> unit) -> unit;
          base : string; ctx : Viewers.context;
          i18n_encoder : string -> string;
          imgmanager : < add_image : Embed.embobject -> unit; .. >;
          pop_action : unit; push_action : (string -> unit) -> unit;
          remove_tag : string -> unit; target : string option; .. > ->
        unit
    end
*)
val init:
        < add_tag : string ->
                    (Htmlfmt.formatter -> Html.tag -> unit) ->
                    (Htmlfmt.formatter -> unit) -> unit;
          base : string; ctx : Viewers.context;
          i18n_encoder : string -> string;
          imgmanager : < add_image : <Cap.network> -> Embed.obj -> unit; .. >;
          pop_action : unit;
          push_action : (string -> unit) -> unit;
          remove_tag : string -> unit; target : string option; .. > ->
        unit
```

F.8.20 display/html_form.ml

⟨display/html_form.ml 383⟩≡

```
(*
 * Level 2 stuff (forms)
 *)

(* The behaviour of a form *)

class behaviour (base, formtag, deftarget, i18n_encoder) =
  object (self)
    val mutable (*private*) elem_value
        : (Htmlfmt.input_kind * (unit -> (string * string) list)) list =
      []

    val mutable (*private*) elem_reset : (unit -> unit) list = []

    val (*private*) fmethod =
      String.uppercase_ascii (Html.get_attribute formtag "method")

    val (*private*) encoding = Html.get_attribute formtag "enctype"

    (* val i18n_encoder = i18n_encoder *)
    val (*private*) action =
      try Html.get_attribute formtag "action" with
      | Not_found -> base

    (* val base = base *)
    val (*private*) h_params =
      try [ ("target", Html.get_attribute formtag "target") ] with
      | Not_found -> (
          match deftarget with
          | Some s -> [ ("target", s) ]
          | None -> [])

    val mutable (*private*) entries = 0 (* number of text entries *)

    (* Contribute a value to the form *)
    method add_get kind f =
      elem_value <- (kind, f) :: elem_value;
      if kind = EntryInput then entries <- entries + 1

    (* How to reset the element *)
    method add_reset f = elem_reset <- f :: elem_reset

    (* The link for a given submit activation *)
    method submit (l : (string * string) list) : Hyper.link =
      let values =
        List.flatten
          (List.map
             (function
               | _, f -> f ())
             elem_value)
      in
      let values = l @ values in
      (* These values must be encoded in the same Kanji code of the source here,
       * if it is Japanese mode. And the difficulty is the Kanji code of the
       * document is usually lazily determined. --- JPF
       *)
      let values_i18n = List.map (fun (t, v) -> (t, i18n_encoder v)) values in
      let evalues = Urlenc.form_encode (List.rev values_i18n) in
      match fmethod with
      | "POST" ->
          Hyper.
            {
              h_uri = action;
              h_context = Some base;
              h_method = POST evalues;
              h_params;
            }
      | _ ->
          let uri =
            let l = String.length action in
            if l = 0 then Printf.sprintf "?%s" evalues
            else if action.[l - 1] = '?' then action ^ evalues
            else Printf.sprintf "%s?%s" action evalues
          in
          { h_uri = uri; h_context = Some base; h_method = GET; h_params }

      (* Submit if only one entry in the form. This may not be the proper test. *)
      method single_submit = if entries = 1 then Some (self#submit []) else None

      (* Resetting the form *)
      method reset = List.iter (fun f -> f ()) elem_reset
    end

(*
module Make(FormDisplay : FormDisplay) =
 struct
  open FormDisplay
*)
module FormDisplay = Form
(*
 * <!ELEMENT FORM - - %body.content -(FORM)>
 * <!ATTLIST FORM
 *         action %URL #REQUIRED -- server-side form handler --
 *         method (%HTTP-Method) GET -- see HTTP specification --
 *         enctype %Content-Type; "application/x-www-form-urlencoded"
 *         >
 *)

let init mach =
  mach#add_tag "form"
    (fun _fo tform ->
      let behav =
        new behaviour (mach#base, tform, mach#target, mach#i18n_encoder)
      in
      let fm = FormDisplay.create mach#base behav mach#ctx in

      (* 8.1.2 Input Field : INPUT *)
      let open_input (fo : Htmlfmt.formatter) t =
        let inputtype = String.uppercase_ascii (Html.get_attribute t "type") in
        (* Special case for hidden, since there is no formatting *)
        (* HTML 3.2 doesn't specify that NAME and VALUE are required, but
           this is stupid *)
        if inputtype = "HIDDEN" then begin
          try
            let name = Html.get_attribute t "name" in
            let v = Html.get_attribute t "value" in
            behav#add_get OtherInput (fun () -> [ (name, v) ])
          with
          | Not_found ->
              raise (Html.Invalid_Html "missing NAME or VALUE in input HIDDEN")
        end
        else
          (* Other cases *)
          let fr =
            fo.create_embedded (Html.get_attribute t "align") None None
          in
          match inputtype with
          | "TEXT"
          | "PASSWORD" ->
              fm.text_input fr t
        | "CHECKBOX" -> fm.checkbox_input fr t
        | "RADIO" -> fm.radio_input fr t
        | "IMAGE" ->
            let caps = Cap.network_caps_UNSAFE () in
            mach#imgmanager#add_image caps (fm.image_input fr t)
        | "SUBMIT" -> fm.submit_input fr t
        | "RESET" -> fm.reset_input fr t
        (* TODO: file *)
        | s -> raise (Html.Invalid_Html ("Invalid INPUT TYPE=" ^ s))
    in
    mach#add_tag "input" open_input (fun _ -> ());

    (* 8.1.3 Selection : SELECT *)
    (* the /SELECT does all the job, so we have to transmit the info ! *)
    let options = ref [] (* the components from which to select *)
    and tselect = ref Html.{ tag_name = "select"; attributes = [] } in
    let open_select _fo t =
      options := [];
      tselect := t;
      mach#add_tag "option"
        (fun _fo tag ->
          mach#push_action (fun s ->
              let s = Html.beautify2 s in
              (* the val is by default the "content" of the tag *)
              let va =
                try Html.get_attribute tag "value" with
                | Not_found -> s
              in
              options :=
                (va, s, Html.has_attribute tag "selected") :: !options))
        (fun _ -> mach#pop_action)
    and close_select (fo : Htmlfmt.formatter) =
      mach#remove_tag "option";
      let fr =
        fo.create_embedded (Html.get_attribute !tselect "align") None None
      in
      fm.select fr (List.rev !options) !tselect
    in

    mach#add_tag "select" open_select close_select;

    (* 8.1.4 Text Area: TEXTAREA *)
    let textarea_initial = Ebuffer.create 128
    and ttextarea = ref Html.{ tag_name = "textarea"; attributes = [] } in
    let open_textarea _fo tag =
      ttextarea := tag;
      Ebuffer.reset textarea_initial;
      mach#push_action (fun s -> Ebuffer.output_string textarea_initial s)
    and close_textarea (fo : Htmlfmt.formatter) =
      mach#pop_action;
      let _nameTODO = Html.get_attribute !ttextarea "name" in
      let fr =
        fo.create_embedded (Html.get_attribute !ttextarea "align") None None
      in
      fm.textarea fr (Ebuffer.get textarea_initial) !ttextarea
    in

    mach#add_tag "textarea" open_textarea close_textarea)
(fun _fo -> [ "input"; "select"; "textarea" ] |> List.iter mach#remove_tag)
```

```
  (*end *)
```

F.8.21 display/htmlw.mli

⟨*signature* Htmlw.frames_as_links 387a⟩≡ (387f)
```
  val frames_as_links : bool ref
```

⟨*signature* Htmlw.pscrolling 387b⟩≡ (387f)
```
  val pscrolling : bool ref
```

⟨*signature* Htmlw.ignore_meta_charset 387c⟩≡ (387f)
```
  val ignore_meta_charset : bool ref
```

⟨*signature* Htmlw.progress_report 387d⟩≡ (387f)
```
  val progress_report :
      Widget.widget -> Viewers.context -> Widget.widget * Scheduler.progress_func
```

⟨*signature* Htmlw.html_head_ui 387e⟩≡ (387f)
```
  val html_head_ui :
      string list -> (unit -> unit) -> bool ref -> Widget.widget ->
        Viewers.context ->
      Widget.widget * (string -> unit) * (string -> Hyper.link -> unit) *
      (string -> string -> unit) * ((Widget.widget -> unit) -> unit)
  (* [html_head_ui headers redisplay scrollmode top ctx]
     returns
     hgroup, set_title, add_link, add_header, add_extra_header
  *)
```

⟨display/htmlw.mli 387f⟩≡
```

  ⟨signature Htmlw.frames_as_links 387a⟩
  ⟨signature Htmlw.pscrolling 387b⟩
  ⟨signature Htmlw.ignore_meta_charset 387c⟩

  class  virtual viewer_globs : (Viewers.context * Document.handle) -> object
    (* val ctx : Viewers.context *)
    val mutable dh : Document.handle
    val did : Document.id
    method ctx : Viewers.context
    method dh : Document.handle
    method did : Document.id
  end
  (*pad: seems mostly boilerplate getters of params *)

  ⟨signature Htmlw.progress_report 387d⟩

  ⟨signature Htmlw.html_head_ui 387e⟩
  ⟨signature Htmlw.display_html 122a⟩
```

F.8.22 display/htmlw.ml

⟨*constant* Htmlw.pscrolling 387g⟩≡ (391b)
```
  let pscrolling = ref false
```

⟨*constant* Htmlw.ignore_meta_charset 387h⟩≡ (391b)
```
  let ignore_meta_charset = ref false
```

⟨*constant* Htmlw.scroll_icon 388a⟩≡ (391b)
```
  let scroll_icon ="
  #define mini-scroll-arrows_width 16
  #define mini-scroll-arrows_height 14
  static char mini-scroll-arrows_bits[] = {
   0x80,0x00,0xc0,0x01,0xc0,0x01,0xe0,0x03,0xf0,0x07,0xf0,0x07,0x00,0x00,0x00,
   0x00,0xf0,0x07,0xf0,0x07,0xe0,0x03,0xc0,0x01,0xc0,0x01,0x80,0x00};
  "
```

⟨*constant* Htmlw.scroll_image 388b⟩≡ (391b)
```
  let scroll_image =
    lazy (ImageBitmap(Imagebitmap.create [Data scroll_icon]))
```

⟨*function* Htmlw.html_head_ui 388c⟩≡ (391b)
```
  let html_head_ui headers redisplay pscroll top (ctx : Viewers.context) =
    (* The frame for all head UI elements *)
    let headgroup = Frame.create_named top "head" [] in
    (* The menubar frame *)
    let bargroup = Frame.create_named headgroup "menubar" [] in
    let titlev = Textvariable.create_temporary headgroup in

    let headersb = Menubutton.create_named bargroup "headers"
        [TextVariable titlev; TextWidth 80]
    in
    let headersm = Menu.create_named headersb "menu" [] in
    Menubutton.configure headersb [Menu headersm];

    (* The link button and menu *)
    let linkb =
      Menubutton.create_named bargroup "links" [Text "Links"; State Disabled] in
    let linkmenu = Menu.create_named linkb "linkmenu" [] in
    Menubutton.configure linkb [Menu linkmenu];

    (* The scroll-mode button *)
    let scrollv = Textvariable.create_temporary bargroup in
    Textvariable.set scrollv
        (if !pscroll then "1" else "0");
    let scrollb =
      Checkbutton.create_named bargroup "smoothScroll"
        [ Variable scrollv; Lazy.force scroll_image;
      IndicatorOn false; Command (fun () ->
        match Textvariable.get scrollv with
          "1" -> pscroll := true; redisplay()
        | _ -> pscroll := false; redisplay())]
    in
    (* bargroup IN THIS ORDER -- RESIZING *)
    pack [linkb][Side Side_Right];
    pack [scrollb][Side Side_Right; Fill Fill_Y];
    pack [headersb][Side Side_Left; Fill Fill_X; Expand true];
    pack [bargroup][Side Side_Top; Fill Fill_X];

    let set_title t =
      let tl = Winfo.toplevel top
      and title = s_ "MMM Browser@%s" t in
      if Widget.known_class tl = "toplevel" then
        (Wm.title_set tl title; Wm.iconname_set tl title);
      Textvariable.set titlev t

    and add_link title hlink =
      Menubutton.configure linkb [State Normal];
      Menu.add_command linkmenu [ Label title;
                  Command (fun () -> ctx#goto hlink)]

    in

    (* Extra headers: META tags should be parsed by *servers*, not by clients.
       TODO: find interface so we can export this feature to applets/modules
       *)
    let sep_added = ref false in
    let add_header h v =
      if not !sep_added then (sep_added := true; Menu.add_separator headersm);
      let txt = sprintf "%s: %s" h v in
      match String.lowercase_ascii h with
        "refresh" ->
         begin try
            let pos = String.index v ';'
            and pos2 = String.index v '=' in
            let _delay = int_of_string (String.sub v 0 pos)
            and url = String.sub v (pos2+1) (String.length v - pos2 - 1) in
        Menu.add_command headersm
          [Label txt;
            Command (fun () -> ctx#goto Hyper.{
              h_uri = url;
              h_context = Some (Url.string_of ctx#base.document_url);
              h_method = GET;
              h_params = []})]
      with Not_found | Failure "int_of_string" -> ()
      end
      | _ -> Menu.add_command headersm [Label txt]

    in
    (* the head menu is a good place to put some other information and ui *)
    (* you must be sure that this function is used after all the head ui
     * stuff using add_header are finished.
     *)
    let sep_extra_added = ref false in
    let add_extra_header f =
      if not !sep_extra_added then
         (sep_extra_added := true; Menu.add_separator headersm);
      f headersm
    in

    set_title (Url.string_of ctx#base.document_url);
    List.iter (function h -> Menu.add_command headersm [Label h])
      (List.rev headers);

    headgroup, set_title, add_link, add_header, add_extra_header
```

⟨*function* Htmlw.ignore_open 389a⟩≡ (391b)

```
  (*
   * Extend a display machine to interpret HEAD elements with
   * an influence on the HEAD ui display.
   * NOTE: some other HEAD elements interpretation *must* be done
   * even if we don't have UI for HEAD (e.g. base)
   *)
  let ignore_open _ _ = ()
```

⟨*function* Htmlw.ignore_close 389b⟩≡ (391b)

```
  let ignore_close _ = ()
```

⟨*function* Htmlw.head_hook 390⟩≡ (391b)

```
  let head_hook (headgroup,set_title,add_link,add_header) mach =
    mach#add_tag "title"
      (fun _fo _t ->
        mach#push_action
      (fun s ->
        set_title (Html.beautify2 s);
        mach#pop_action))
      ignore_close;

    mach#add_tag "isindex"
      (fun _fo tag ->
        let prompt = Html.get_attribute tag "prompt" in
        let action s =
           mach#ctx#goto Hyper.{ h_uri = "?" ^ Urlenc.encode s;
                  h_context = Some mach#base;
                  h_method = GET;
                  h_params = []} in
        let f,_e = Frx_entry.new_label_entry headgroup prompt action in
        pack [f] [Fill Fill_X])
      ignore_close;

    mach#add_tag "link"
      (fun _fo tag ->
        try
          let href = Html.get_attribute tag "href" in
      let name =
        try Html.get_attribute tag "title"
        with Not_found ->
          try Html.get_attribute tag "rel"
          with Not_found ->
            try Html.get_attribute tag "rev"
            with Not_found -> href in
      let h_params =
        try ["target", Html.get_attribute tag "target"]
        with
          Not_found ->
            match mach#target with
          Some s -> ["target", s]
            | None -> []
          in
          add_link name Hyper.{ h_uri = href; h_context = Some mach#base;
              h_method = GET; h_params = h_params}
        with
      Not_found -> () (* no href *))
      ignore_close;

    begin
      let old =
        try
      let oldo, _c = mach#get_tag "meta" in oldo
        with
      Not_found -> ignore_open in
      mach#add_tag "meta"
        (fun fo tag ->
         try
        old fo tag;
        add_header
          (Html.get_attribute tag "http-equiv")
          (Html.get_attribute tag "content")
        with Not_found -> ())
       ignore_close;
    end;

    (* Non standard extensions *)
    if !frames_as_links  then
      mach#add_tag "frame"
        (fun _fo tag ->
      try
          let src = Html.get_attribute tag "src" in
          let name =
         sprintf "Frame %s"
           (try Html.get_attribute tag "name"
           with Not_found -> "unnamed")
        in
        add_link name { h_uri = src; h_context = Some mach#base;
                h_method = GET; h_params = []}
      with
        Not_found -> () (* no src *))
        ignore_close
```

⟨*function* Htmlw.embedded_html 391a⟩≡ (391b)

```
  (* TODO: we should be able to share the imgmanager, but I don't see
   *  where we can get it from (except by adding something in ctx)
   *)
  let embedded_viewer =
   fun mediapars top (ctx : Viewers.context) (doc : Document.t) ->
    let imgmanager = Imgload.create() in
    let dh = Decoders.insert (Cache.make_embed_handle doc) in
    let ctx = ctx#in_embed dh.document_id in
    let disp = new display_html (top,ctx,mediapars,imgmanager,dh) in
    disp#init false;
    pack [disp#di_widget][Expand true; Fill Fill_Both];
    let caps = Cap.network_caps_UNSAFE () in
    (* set for events *)
    Frx_synth.bind disp#di_widget "load_images"
      (fun _top -> disp#di_load_images);
    Frx_synth.bind disp#di_widget "update"
      (fun _ -> Embed.update caps top ctx doc (fun () -> disp#di_update))
```

⟨display/htmlw.ml 391b⟩≡

```
  open I18n
  open Tk

  (*****************************************************************************)
  (* Prelude *)
  (*****************************************************************************)

  (*****************************************************************************)
  (* Globals *)
  (*****************************************************************************)
```

  ⟨*constant* Htmlw.frames_as_links 158b⟩
  ⟨*constant* Htmlw.pscrolling 387g⟩
  ⟨*constant* Htmlw.ignore_meta_charset 387h⟩

```
  (*****************************************************************************)
  (* Helpers *)
  (*****************************************************************************)
```

```
⟨constant Htmlw.scroll_icon 388a⟩

⟨constant Htmlw.scroll_image 388b⟩

⟨module Htmlw.F 123g⟩


⟨function Htmlw.progress_report 205d⟩

⟨function Htmlw.html_head_ui 388c⟩


⟨function Htmlw.ignore_open 389a⟩
⟨function Htmlw.ignore_close 389b⟩

⟨function Htmlw.head_hook 390⟩

(* This class only defines globals *)
class  virtual viewer_globs ((ctx : Viewers.context),
                (dh' : Document.handle)) =
 object

  (* copy params *)
  (* val ctx = ctx *)
  method ctx = ctx

  val mutable dh = dh'
  method dh = dh

  val did = dh'.document_id
  method did = did
end

(* We still need dh at construction for the definition of feed_red *)
class  virtual html_parse ((dh : Document.handle)) =
 object (self)

  method virtual dh : Document.handle
  method virtual set_progress: int option -> int -> unit

  (* red tape for progress report *)
  val mutable red = 0
  val mutable size =
    try Some (Http_headers.contentlength dh.dh_headers)
    with Not_found -> None (* duh *)
  val mutable feed_read = new Charset.read_i18n (fun _s _o _n -> 0)

  val mutable (*private*) lexbuf = Lexing.from_string "" (* duh *)
  method lexbuf = lexbuf

  val mutable lexer = Html_eval.sgml_lexer !Dtd.current

  (* Japanese parse configuration *)
  val jpn_config = Charset.default_config ()

  method parse_init =
    red <- 0;
    feed_read <-
       Charset.create_read_native self#dh.document_feed.feed_read;
    (* Q: do we need to restart a new sgml_lexer ? *)
    lexer <- Html_eval.sgml_lexer !Dtd.current;
    lexbuf <-
       Lexing.from_function (fun buf n ->
     let r = feed_read#read buf 0 n in
     red <- red + r;
     self#set_progress size red;
     r)
end

class  virtual html_body () =
 object (self)

  method virtual mach : Html_disp.machine
  method virtual ctx: Viewers.context
  method virtual frame : Widget.widget

  val current_scroll_mode = ref !pscrolling
  method current_scroll_mode = current_scroll_mode

  (* hack for frames on incorrect html *)
  val mutable body_frame = None
  method body_init full =
    (* Install a tag handler for body that will actually create the
       a formatter and install it *)
    let body_formatter = ref None in
    self#mach#add_tag "body"
      (fun _fo t ->
    match !body_formatter with
    | None -> (* it's the first body *)
        let format, fhtml =
          self#mach#create_formatter
           (if full then (TopFormatter !current_scroll_mode)
           else FrameFormatter self#ctx#params)
           self#frame
        in
        self#mach#push_formatter format;
        body_formatter := Some format;
        body_frame <- Some fhtml;
        pack [fhtml][Side Side_Left; Expand true; Fill Fill_Both];
         List.iter (function
          | "bgcolor", color ->
             format.set_defaults "background" [BgColor color]
          | "text", color ->
          format.set_defaults "foreground" [FgColor color]
          | "link", color ->
          format.set_defaults "link" [FgColor color]
          | "alink", color ->
          format.set_defaults "alink" [FgColor color]
          | "vlink", color ->
          format.set_defaults "vlink" [FgColor color]
          | _,_ -> ())
             t.attributes
    | Some f -> (* multiple body... *)
        self#mach#push_formatter f
     )
      (fun _t ->
    ignore self#mach#pop_formatter)
end

(* geek stuff *)
```

```
class  virtual bored () =
 object (self)
  method virtual ctx: Viewers.context
  method virtual mach: F.machine

  method bored_init hgbas =
    let bored =
      Resource.get Widget.default_toplevel "bored" "bored" = "yes"
    || begin try
       ignore (
         Str.search_forward (Str.regexp_case_fold "sandra") self#mach#base 0);
       Resource.add "*bored" "yes" Interactive;
       true
    with Not_found -> false
    end in
    if bored then begin
      let b = Button.create hgbas [
       Text "\182"; BorderWidth (Pixels 0);
       Command (fun () ->
      self#ctx#goto
       (Hyper.default_link
         "http://www.columbiatristar.co.uk/the_net/contents.html")
      )] in
      let l  = Winfo.children hgbas in
      pack [b][After (List.hd l); Side Side_Right]
    end

end

(*****************************************************************************)
(* Display_info *)
(*****************************************************************************)

⟨class Htmlw.display_html 122c⟩

(*****************************************************************************)
(* Entry point *)
(*****************************************************************************)

⟨function Htmlw.display_html 116a⟩

⟨function Htmlw.embedded_html 391a⟩

⟨toplevel Htmlw._1 115e⟩
⟨toplevel Htmlw._2 161a⟩
```

F.8.23 display/form.mli

⟨*signature* Form.form_bg 394a⟩≡ (395b)

```
val form_bg : string ref
```

⟨*type* Form.t 394b⟩≡ (395b)

```
type t = {
  text_input : Widget.widget -> Html.tag -> unit;
      (* [text_input top tag] *)
  checkbox_input : Widget.widget ->  Html.tag -> unit;
      (* [input top tag] *)
  radio_input : Widget.widget ->  Html.tag -> unit;
      (* [input top tag] *)
```

```
    image_input : Widget.widget ->  Html.tag -> Embed.obj;
        (* [input top tag] *)
    submit_input : Widget.widget ->  Html.tag -> unit;
        (* [input top tag] *)
    reset_input : Widget.widget ->  Html.tag -> unit;
        (* [input top tag] *)
    select : Widget.widget -> (string * string * bool) list -> Html.tag -> unit;
        (* [select top elements tag] *)

    textarea:  Widget.widget -> string -> Html.tag -> unit
        (* [textarea top initial attrs] *)
  }
```

⟨*signature* Form.create 395a⟩≡ (395b)

```
  val create:
    string -> Htmlfmt.form_behaviour -> Viewers.context -> t
```

⟨display/form.mli 395b⟩≡

```
  ⟨signature Form.form_bg 394a⟩

  ⟨type Form.t 394b⟩

  ⟨signature Form.create 395a⟩
```

F.8.24 display/form.ml

⟨*constant* Form.form_bg 395c⟩≡ (401b)

```
  let form_bg = ref "#d9d9d9"
```

⟨type Form.t (./display/form.ml) 395d⟩≡ (401b)

```
  type t = {
    text_input : Widget.widget -> tag -> unit;
        (* [text_input top tag] *)
    checkbox_input : Widget.widget ->  tag -> unit;
        (* [input top tag] *)
    radio_input : Widget.widget ->  tag -> unit;
        (* [input top tag] *)
    image_input : Widget.widget ->  tag -> Embed.obj;
        (* [input top tag] *)
    submit_input : Widget.widget ->  tag -> unit;
        (* [input top tag] *)
    reset_input : Widget.widget ->  tag -> unit;
        (* [input top tag] *)
    select : Widget.widget -> (string * string * bool) list -> tag -> unit;
        (* [select top elements tag] *)
    textarea:  Widget.widget -> string -> tag -> unit
        (* [textarea top initial attrs] *)
    }
```

⟨*function* Form.mapi 395e⟩≡ (401b)

```
  (* mapi (fun n x -> ...) n0 l *)
  let rec mapi f n l =
    match l with
      [] -> []
    | x::l -> let v = f n x in v::(mapi f (succ n) l)
```

⟨*function* Form.focus_link 396a⟩≡ (401b)
```
  (* Do our own Tab/Shift-Tab : don't call it for Text widgets ! *)
  let focus_link prev w =
    match !prev with
      None -> prev := Some w;
    | Some old ->
        bind old [[], KeyPressDetail "Tab"]
         (BindSetBreakable ([], fun _ei -> try Focus.set w with _ -> ()));
        bind w [[Shift], KeyPressDetail "Tab"]
         (BindSetBreakable ([], fun _ei -> try Focus.set old with _ -> ()));
        prev := Some w
```

⟨*function* Form.text_input 396b⟩≡ (401b)
```
  (* TODO: MAXLENGTH *)
  let text_input prev_widget ctx behav top tag =
    try
      let name = get_attribute tag "name"
      and inputtype = get_attribute tag "type" in
      (* Create an entry widget : don't take focus unless we click in it *)
      let e = Entry.create top [ExportSelection false; TakeFocus false;
                      Background (NamedColor !form_bg)] in
      (* Check for size *)
      begin try
       let s = get_attribute tag "size" in
        try Entry.configure e [TextWidth (int_of_string s)]
        with Failure "int_of_string" ->
       Log.f (sprintf "%s not a valid val for SIZE" s)
      with Not_found -> ()
      end;
      (* Check for passwd *)
      if String.lowercase_ascii inputtype = "password" then
        Entry.configure e [Show '*'];
      (* The behaviours *)
      let reset =
        try
      let v = get_attribute tag "value" in
       Entry.insert e End v;
       (fun () -> Entry.delete_range e (Number 0) End;
              Entry.insert e End v)
        with Not_found ->
       (fun () -> Entry.delete_range e (Number 0) End)
      (* spec says we could omit if empty *)
      and get_value () = [name, Entry.get e]
      in
        focus_link prev_widget e;
        pack [e][];
        behav#add_get EntryInput get_value;
        behav#add_reset reset;
       (* single-entry : enter submits the form *)
        Tk.bind e [[], KeyPressDetail "Return"]
           (BindSet ([], (fun _ ->
         match behav#single_submit with
           Some h -> ctx#goto h
         | None -> ())))
    with
      Not_found ->
        raise (Invalid_Html "Missing NAME in TEXT or PASSWORD")
```

⟨*function* Form.checkbox_input 396c⟩≡ (401b)
```
  (* A CHECKBOX input *)
```

```
let checkbox_input prev_widget behav top tag =
  try
    let name = get_attribute tag "name" in
    let v = Textvariable.create_temporary top in  (* variable val is 1/0 *)
    let c = Checkbutton.create top [Variable v; TakeFocus false] in
    let reset =
      if has_attribute tag "checked" then begin
    Checkbutton.select c;
    (fun () -> Checkbutton.select c)
    end
      else (fun () -> Checkbutton.deselect c)
    and get_value =
      let value =
    try get_attribute tag "value"
    with Not_found ->
        (* other browsers seem to use "on". Thanks to Dave Love for pointing
           this out *)
     Log.f "no VALUE given for input CHECKBOX, using \"on\"";
     "on" in
      (* spec says we SHOULD omit when not selected *)
      (fun () ->
     match Textvariable.get v with
       "1" -> [name, value]
     | _ -> [])
    in
      focus_link prev_widget c;
      pack [c][];
      behav#add_get OtherInput get_value;
      behav#add_reset reset
  with
    Not_found ->
      raise (Invalid_Html "Missing NAME in CHECKBOX")
```

⟨*function* Form.radio_input 397⟩≡ (401b)

```
  (* ONLY THE FIRST BUTTON RESET/GET_VALUE IS USED *)
  let radio_input prev_widget behav =
    (* Table of radio names *)
    let radios = Hashtbl.create 17 in
    (fun top tag ->
       try
         let name = get_attribute tag "name" in
         let r = Radiobutton.create top [TakeFocus false]
         and checked = has_attribute tag "checked"
         and va = try get_attribute tag "value" with Not_found -> "on"
         in
         try
       let v, sel = Hashtbl.find radios name in
         (* We already have a radiobutton with this name *)
         Radiobutton.configure r [Variable v; Value va];
         if checked then begin
           Radiobutton.select r; (* select it *)
           sel := r (* store it in table for reset *)
         end;
         (* no need to add behaviour *)
         focus_link prev_widget r;
         pack [r][]
         with
       Not_found ->
         (* this is the first radio button with this name *)
         (* it this thus assumed checked *)
```

```
        let v = Textvariable.create_temporary top in
         Hashtbl.add radios name (v, ref r);
        let get_value () = [name, Textvariable.get v]
        and reset () =
           (* to reset, we must lookup the table *)
          let _, sel = Hashtbl.find radios name in
        Radiobutton.select !sel
        in
        Radiobutton.configure r [Variable v; Value va];
        Radiobutton.select r; (* assume selected *)
        focus_link prev_widget r;
        pack [r][];
        behav#add_get OtherInput get_value;
        behav#add_reset reset
    with
      Not_found ->
    raise (Invalid_Html "Missing NAME in RADIO"))
```

⟨*function* Form.image_input 398a⟩≡ (401b)

```
  (* An IMAGE input
   * Q: no target here ?
   *)
  let image_input _prev_widget ctx base behav top tag =
    try
      let n = get_attribute tag "name" in
      let src = get_attribute tag "src" in
      let alt =
         try get_attribute tag "alt"
         with Not_found -> "[INPUT IMAGE]"
      in
        {
        embed_hlink =
            { h_uri = src; h_context = Some base; h_method = GET; h_params =[] };
        embed_frame = top;
        embed_context = ctx; (* pass as is... *)
        embed_map = FormMap (fun (x, y) ->
                   let subargs =
                                  [sprintf "%s.x" n, string_of_int x;
                                   sprintf "%s.y" n, string_of_int y] in
                     behav#submit subargs);
        embed_alt = alt}
    with
    Not_found ->
      raise (Invalid_Html "missing NAME or SRC in input IMAGE")
```

⟨*function* Form.submit_input 398b⟩≡ (401b)

```
  (* A Submit button *)
  let submit_input _prev_widget ctx behav top tag =
    let l =
      try get_attribute tag "value"
      with Not_found -> s_ "Submit"
    in
    try
      let n = get_attribute tag "name" in
      pack [Button.create top [Text l; TakeFocus false;
           Command (fun () -> ctx#goto (behav#submit [n,l]))]]
           []
    with
      Not_found ->
       (* if name is not present, the button does not contribute a value *)
      pack [Button.create top [Text l; TakeFocus false;
           Command (fun () -> ctx#goto (behav#submit []))]]
       []
```

⟨*function* Form.reset_input 399a⟩≡ (401b)

```
  let reset_input _prev_widget behav top tag =
    let l =
      try get_attribute tag "value"
      with Not_found -> s_ "Reset" in
    let b = Button.create top [Text l; TakeFocus false;
                  Command (fun () -> behav#reset)] in
      pack[b][]
```

⟨*function* Form.select 399b⟩≡ (401b)

```
  (* options is: (val, displayed thing, selected) list *)
  let select prev_widget behav top options tag =
    let name = get_attribute tag "name" in
    let ssize = get_attribute tag "size" in
    let size =
       try int_of_string ssize
       with _ ->
      Log.f (sprintf "%s not a valid val for SIZE" ssize);
      5 in
    let multiple = has_attribute tag "multiple" in
    (* assume 20 vertical pixels per menu item *)
    let fit_vertical n = 20 * n < Winfo.screenheight top in
    if size = 1 && not multiple && fit_vertical (List.length options)
    then begin (* menus larger than screen are bad, use an optionmenu *)
      let vard = Textvariable.create_temporary top   (* var to display *)
      and varv = Textvariable.create_temporary top   (* var for val *)
      in
      let m = Menubutton.create top
           [TextVariable vard; Relief Raised; Anchor Center; TakeFocus false] in
      let mmenu = Menu.create m [TearOff false] in
       Menubutton.configure m [Menu mmenu];
       let initial =
         match options with
       [] -> raise (Invalid_Html ("No OPTION in SELECT"))
         | opt :: _ -> ref opt in
       List.iter (function (v,d,s) as x ->
          Menu.add_command mmenu
              [Label d;
               Command (fun () ->
                     Textvariable.set varv v;
                     Textvariable.set vard d
                    )];
          if s then initial := x
          )
            options;
       let reset () =
         match !initial with
      (v,d,_) ->
           Textvariable.set varv v;
           Textvariable.set vard d
       and get_value () = [name, Textvariable.get varv] in
         reset();
         focus_link prev_widget m;
         pack [m][];
         behav#add_get OtherInput get_value;
         behav#add_reset reset
      end
    else begin (* use a listbox *)
     (* listbox indices start at 0 *)
     (* we must not ExportSelection, otherwise one unique listbox can *)
     (* have a current selection *)
      let nth_entry n l =
        let (v,_,_) = List.nth l n in v in
      let f,lb = Frx_listbox.new_scrollable_listbox top
        [TextHeight size; TextWidth 0; (* automatic size *)
         (if multiple then SelectMode Multiple else SelectMode Single);
         ExportSelection false; Background (NamedColor !form_bg)] in
        Listbox.configure lb [TakeFocus false];
      let initial = ref [] in
      let entries =
       mapi (fun i (_,v,s) ->
             if s then initial := i :: !initial;
             v) 0 options in
        Listbox.insert lb End entries;
      if !initial = [] then initial := [0];
      let reset () =
         Listbox.selection_clear lb (Number 0) End;
         List.iter (fun i ->
           Listbox.selection_set lb (Number i)(Number i))
             !initial
      and get_value () =
         List.map (function
            Number n -> name, nth_entry n options
          | _ -> name, nth_entry 0 options (* fatal error ! *))
         (Listbox.curselection lb)
      in
        reset ();
        focus_link prev_widget f;
        pack [f][];
        behav#add_reset reset;
        behav#add_get OtherInput get_value
      end
```

⟨*function* Form.textarea 400⟩≡ (401b)

```
  let textarea _prev_widget behav top initial tag =
    try
      let name = get_attribute tag "name" in
      let f,t =
         Frx_text.new_scrollable_text top
             [ExportSelection false; TakeFocus false] false in
      Text.configure t [Background (NamedColor !form_bg)];
      begin try
        let w = get_attribute tag "cols" in
        try Text.configure t [TextWidth (int_of_string w)]
        with Failure "int_of_string" ->
      Log.f (sprintf "%s not a valid val for COLS" w)
      with Not_found -> ()
      end;
      begin try
        let h = get_attribute tag "rows" in
        try Text.configure t [TextHeight (int_of_string h)]
        with Failure "int_of_string" ->
      Log.f (sprintf "%s not a valid val for ROWS" h)
      with Not_found -> ()
      end;
      Text.insert t Frx_text.textEnd initial [];
      let reset () =
        Text.delete t (TextIndex(LineChar(0,0), [])) Frx_text.textEnd;
        Text.insert t Frx_text.textEnd initial []
      and get_value () =
        [name, Text.get t (TextIndex(LineChar(0,0), []))
             (TextIndex(End, [CharOffset (-1)]))]
      in
         pack [f][];
         behav#add_reset reset;
         behav#add_get EntryInput get_value
    with
      Not_found -> raise (Invalid_Html "Missing NAME in TEXTAREA")
```

⟨*function* Form.create 401a⟩≡ (401b)

```
  let create base behav ctx =
    let prev_widget = ref None in
    { text_input = text_input prev_widget ctx behav;
      checkbox_input = checkbox_input prev_widget behav;
      radio_input = radio_input prev_widget behav;
      image_input = image_input prev_widget ctx base behav;
      submit_input = submit_input prev_widget ctx behav;
      reset_input = reset_input prev_widget behav;
      select = select prev_widget behav;
      textarea = textarea prev_widget behav
    }
```

⟨display/form.ml 401b⟩≡

```
  (* Tk based FormDisplay  *)
  open I18n
  open Printf
  open Tk
  open Hyper

  open Html
  open Htmlfmt
  open Maps
  open Embed


  ⟨constant Form.form_bg 395c⟩

  ⟨type Form.t (./display/form.ml) 395d⟩

  (* Most of the widgets created here are created with [TakeFocus false];
   * The reason is that otherwise, in "focus follows mouse" mode, when scrolling
   * a text widget containing form elements, the mouse may come over one of
   * these elements; it would then set the focus to the element, and break
   * further scrolling. Instead, with [TakeFocus false], the user has to
   * click explicitly in the widgets (especially entries and text) in order
   * to fill them.
   * However, we do attempt to re-implement the Tab/Shift-Tab system that we
   * inevitably broke by setting [TakeFocus false].
   * Phew.
   *)

  ⟨function Form.mapi 395e⟩

  ⟨function Form.focus_link 396a⟩

  (* A TEXT or PASSWORD input *)
  ⟨function Form.text_input 396b⟩

  ⟨function Form.checkbox_input 396c⟩

  (* A RADIO input *)
  ⟨function Form.radio_input 397⟩

  ⟨function Form.image_input 398a⟩

  ⟨function Form.submit_input 398b⟩

  ⟨function Form.reset_input 399a⟩

   (* TODO: FILE (RFC 1867) *)

  (* A SELECT list *)
  ⟨function Form.select 399b⟩

  ⟨function Form.textarea 400⟩

  ⟨function Form.create 401a⟩
```

F.8.25 display/html_table.mli

```
⟨display/html_table.mli 402a⟩≡
  (*
  module Make :
    functor (TableDisplay : Htmlfmt.TableDisplay) ->
      sig
        val init :
          < add_tag : string ->
                      (Htmlfmt.formatter -> Html.tag -> unit) ->
                      (Htmlfmt.formatter -> unit) -> unit;
            create_formatter : Htmlfmt.formatterSpec ->
                               Widget.widget -> 'a * Widget.widget;
            pop_formatter : 'b; push_formatter : 'a -> 'c;
            remove_tag : string -> unit; .. > ->
          unit
      end
  *)
  val init:
          < add_tag : string ->
                      (Htmlfmt.formatter -> Html.tag -> unit) ->
                      (Htmlfmt.formatter -> unit) -> unit;
            create_formatter : Htmlfmt.formatterSpec ->
                               Widget.widget -> 'a * Widget.widget;
            pop_formatter : 'b; push_formatter : 'a -> unit;
            remove_tag : string -> unit; .. > ->
          unit
```

F.8.26 display/html_table.ml

```
⟨display/html_table.ml 402b⟩≡
  open Html
  open Htmlfmt

  (*
  HTML Tables are defined by RFC1942, e.g.
    <URL:ftp://ds.internic.net/rfc/rfc1942.txt>

  This code *assumes* that minimisation rules are used for
  cells (td and th) and for rows.
   *)

  (*
  module Make (TableDisplay: TableDisplay) =
  struct
  open TableDisplay
  *)
  module TableDisplay = Table
  open Table

  let init mach =

    (* Tables may be nested, so we need to remember *)
    let table_stack = ref ([] : TableDisplay.t list) in

    (* Access to the stack *)
    let tm () = match !table_stack with
        tm::_ -> tm
      | [] -> raise (Invalid_Html "Table element outside <TABLE></TABLE>")
    and pop_table () = match !table_stack with
      | _tm::l -> table_stack := l
      | [] -> raise (Invalid_Html "Unmatched </TABLE>")
    and push_table tm = table_stack := tm :: !table_stack
    and _is_nested () =
      match !table_stack with
        [] -> false
      | _ -> true
    in

    (* Layout information : the current constraint width *)

    let widths = ref [] in (* because tables are nested *)
    let current_width () =
      match !widths with
        [] -> TopWidth
      | x::_l -> x
    and push_width w =
      widths := w :: !widths
    and pop_width () =
      match !widths with
        [] -> ()
      | _x::l -> widths := l
    in
    (* <TABLE> starts a table *)
    let open_table fo t =
      fo.new_paragraph();
      (* Create the widget for embedding this table *)
      let fr = fo.create_embedded "" None None in
      (* And the table manager *)
      let tm = TableDisplay.create fr t (current_width()) in
      (* push the table on the stack *)
      push_table tm;
      (* define behavior of other tags *)
      (* align/valign attributes *)
      let current_row_align = ref None
      and current_row_valign = ref None
      in
      let change_aligns t =
        current_row_align :=
       (try Some (String.lowercase_ascii (get_attribute t "align"))
        with Not_found -> None);
        current_row_valign :=
       (try Some (String.lowercase_ascii (get_attribute t "valign"))
        with Not_found -> None)
      in

     let cell_aligns attrs =
       begin try Some (String.lowercase_ascii (get_attribute attrs "align"))
         with Not_found -> !current_row_align
       end,
       begin try Some (String.lowercase_ascii (get_attribute attrs "valign"))
         with Not_found -> !current_row_valign
       end
     in
      (* <TR> : starts a row *)
      let open_tr _fo t = change_aligns t; tm.open_row t
      and close_tr _fo = tm.close_row() in
      mach#add_tag "tr" open_tr close_tr;

      (* A new cell *)
      let open_cell kind _fo t =
        let align,_valign = cell_aligns t in
        let align = match align with
      Some align -> align
        | None -> match kind with
        HeaderCell -> "center"
      | DataCell -> "left"
        in
        (* Create a new formatter, given as parent the table widget *)
        let formatter, tcell =
      mach#create_formatter NestedFormatter tm.table_master in
      mach#push_formatter formatter;
      push_width (tm.new_cell kind t tcell align)

      and close_cell fo =
        (* fo is the formatter that was open for *this* cell *)
        fo.flush();
        (* pop it *)
        mach#pop_formatter |> ignore;
        pop_width()
      in
      mach#add_tag "th" (open_cell HeaderCell) close_cell;
      mach#add_tag "td" (open_cell DataCell) close_cell;
      mach#add_tag "col" (fun _fo t -> tm.add_col t) (fun _ -> ());

    and close_table fo =
      (* close the table manager *)
      (tm()).close_table();
      pop_table();
      fo.close_paragraph ();
       (* NOTE: this is the correct fo only if minimisation were applied
          and the correct current formatter is passed to close table
        *)
      (* remove tags *)
      List.iter mach#remove_tag ["tr";"th";"td";"col"];
```

```
    in

    mach#add_tag "table" open_table close_table;

  (* end *)
```

F.8.27 display/styles.mli

⟨*signature* Styles.init 405a⟩≡ (405e)
```
  val init : string -> string -> unit
      (* [init family slant] *)
```

⟨*signature* Styles.set_font 405b⟩≡ (405e)
```
  val set_font : string -> Fonts.fontAttrs -> unit
      (* [set_font symbolic_name attrs] *)
```

⟨*signature* Styles.get_font 405c⟩≡ (405e)
```
  val get_font : string -> Fonts.fontAttrs
      (* [get_font symbolic_name] *)
```

⟨*signature* Styles.get 405d⟩≡ (405e)
```
  (*
   * Retrieves graphical attributes for a given font
   *)

  val get : string -> Htmlfmt.gattr list
```

⟨display/styles.mli 405e⟩≡
```
  (*
   * Definition of attributes of symbolic fonts (font-modifiers)
   *)

  ⟨signature Styles.init 405a⟩

  ⟨signature Styles.set_font 405b⟩
  ⟨signature Styles.get_font 405c⟩

  ⟨signature Styles.get 405d⟩
```

F.8.28 display/styles.ml

⟨*constant* Styles.fonttable 405f⟩≡ (407a)
```
  (* Definition of font attributes *)
  let fonttable = (Hashtbl.create 37 : (string, fontAttrs) Hashtbl.t)
```

⟨*constant* Styles.get_font 405g⟩≡ (407a)
```
  let get_font =  Hashtbl.find fonttable
```

⟨*function* Styles.set_font 405h⟩≡ (407a)
```
  let set_font name attrs =
    Hashtbl.remove fonttable name;
    Hashtbl.add fonttable name attrs;
    if name = "default" then
      Fonts.default := Fonts.merge !Fonts.default attrs
```

⟨*constant* Styles.table 406a⟩≡ (407a)
```
(*
 * Graphical attributes for a given symbolic name
 * TODO: to support a notion of style sheet, this table should be
 * specific to each display machine, and should define all the properties
 * of the style sheet display model
 *)
let table = (Hashtbl.create 37 : (string, gattr list) Hashtbl.t)
```

⟨*function* Styles.get 406b⟩≡ (407a)
```
(* Merge font attributes and other attributes *)
let get s =
  let fontattrs =
   try Hashtbl.find fonttable s with Not_found -> []
  and otherattrs =
   try Hashtbl.find table s with Not_found -> []
  in
  let attrs =  List.map (fun fi -> Font fi) fontattrs @ otherattrs
  in
   if attrs = [] then raise Not_found else attrs
```

⟨*function* Styles.define_style 406c⟩≡ (407a)
```
let define_style name attrs =
  Hashtbl.remove table name;
  Hashtbl.add table name attrs
```

⟨*function* Styles.init 406d⟩≡ (407a)
```
let init family slant =
  Hashtbl.clear fonttable;
  Hashtbl.clear table;
  (* font initialisation is moot if we have preferences,
     but just in case (no preference file at all), we keep it*)
  List.iter (function (name,attrs) -> set_font name attrs)
    [ "default",  [Family family;  Weight "medium"; Slant "r";  FontIndex 3];
      "header1", [Family family;  Weight "bold"; Slant "r"; FontIndex 7];
      "header2", [Family family;  Weight "bold"; Slant "r"; FontIndex 6];
      "header3", [Family family;  Weight "medium"; Slant slant; FontIndex 5];
      "header4", [Family family;  Weight "bold"; Slant "r"; FontIndex 4];
      "header5", [Family family;  Weight "medium"; Slant slant; FontIndex 4];
      "header6", [Family family;  Weight "bold"; Slant "r"; FontIndex 4];
      "bold", [ Weight "bold"];
      "italic", [ Slant slant];
      (* should be a fixed font. Since we have newlines, spacing should be 0 *)
      "verbatim", [Family "courier"];
      "fixed", [Family "courier"]
    ];
  List.iter (function (name,attrs) -> define_style name attrs)
    [ "default", [Justification "center"; Spacing 2];
      "verbatim", [Spacing 1];
      "header1", [Justification "center"; Spacing 20];
      "header2", [Justification "center"; Spacing 10];
      "header3", [Justification "left"; Spacing 10];
      "header4", [Justification "left"; Spacing 5];
      "header5", [Justification "left"];
      "header6", [Justification "left"]
    ]
```

⟨*toplevel* Styles._1 406e⟩≡ (407a)
```
let _ = init "helvetica" "o"
```

⟨display/styles.ml 407a⟩≡
```
  (* Styles are common display attributes *)
  open Htmlfmt
  open Fonts


  ⟨constant Styles.fonttable 405f⟩

  ⟨constant Styles.get_font 405g⟩
  ⟨function Styles.set_font 405h⟩


  ⟨constant Styles.table 406a⟩

  ⟨function Styles.get 406b⟩

  ⟨function Styles.define_style 406c⟩

  ⟨function Styles.init 406d⟩


  ⟨toplevel Styles._1 406e⟩
```

F.8.29 display/table.mli

⟨*signature* Table.debug 407b⟩≡ (408a)
```
  (* TABLES *)

  val debug : bool ref
```

⟨*signature* Table.strict_32 407c⟩≡ (408a)
```
  val strict_32 : bool ref
```

⟨*type* Table.cell_type 407d⟩≡ (408a)
```
  type cell_type = HeaderCell | DataCell
```

⟨*type* Table.t 407e⟩≡ (408a)
```
  type t = {
    table_master : Widget.widget;
    add_col : Html.tag -> unit;
    open_row : Html.tag -> unit;
    close_row : unit -> unit;
    close_table : unit -> unit;
    new_cell : cell_type -> Html.tag -> Widget.widget -> string -> Htmlfmt.width_constraint;
    bound : unit -> bool
    }
```

⟨*signature* Table.create 407f⟩≡ (408a)
```
  val create : Widget.widget -> Html.tag -> Htmlfmt.width_constraint -> t
```

⟨*signature* Table.topwidth 407g⟩≡ (408a)
```
  val topwidth : Widget.widget -> int
```

```
⟨display/table.mli 408a⟩≡
  ⟨signature Table.debug 407b⟩
  ⟨signature Table.strict_32 407c⟩

  ⟨type Table.cell_type 407d⟩

  ⟨type Table.t 407e⟩

  ⟨signature Table.create 407f⟩

  ⟨signature Table.topwidth 407g⟩
```

F.8.30 display/table.ml

```
⟨constant Table.debug 408b⟩≡                                              (415)
  (* Table support using the grid manager and a gross hack to obtain
     resizing of a text widget to show its entire content.

   * Notes:
    1  we must keep geometry propagation on the grid, otherwise we'll never
       get vertical resizing
    2  the same is valid for each cell (frame around text)
    3  the spec says that the table should grow to fit its contents. However,
       this is ambiguous because in practice we must limit the width to the
       currently displayed page width.
       Having geometry propagation turned on, and letting all cells grow will
       of course keep the grid growing...
       Thus we have to set a maximum width for each cell.
       For text cells, we have to put a limit on their "automatic" horizontal
       resizing. When the limit is reached, we switch to vertical resizing,
       resetting wordwrap if allowed.
   *)

  let debug = ref false

⟨constant Table.strict_32 408c⟩≡                                          (415)
  let strict_32 = ref true
      (* in this mode, we ignore WIDTH of TD defined with %
         This is also better for pages written for MSIE where you find
         either TD WIDTH=100% or TD WIDTH=NN
         *)

⟨type Table.cell_type (display/table.ml) 408d⟩≡                           (415)
  (* a manager for a single TABLE *)
  type cell_type = HeaderCell | DataCell

⟨type Table.t (display/table.ml) 408e⟩≡                                   (415)
  type t = {
    table_master : Widget.widget;
    add_col : Html.tag -> unit;
    open_row : Html.tag -> unit;
    close_row : unit -> unit;
    close_table : unit -> unit;
    new_cell : cell_type -> Html.tag -> Widget.widget -> string -> width_constraint;
    bound : unit -> bool
    }
```

⟨*type* Table.table 409a⟩≡ (415)
```
  (* Internal structure of tables *)
  type table = {
    master_widget : Widget.widget;
    _width : length;
    mutable slaves :
        (Widget.widget * (int*int*int*int*width_constraint*length*string)) list;
    mutable cur_col : int;
    mutable cur_row : int;
    mutable slots : int array;
    mutable cols : int option list;
    cellpadding : int;
    cellspacing : int
    }
```

⟨*function* Table.topwidth 409b⟩≡ (415)
```
  (* Get up to the widget that has HFrame class, or to toplevel *)
  let topwidth wid =
    let f = ref wid in
    try
      while true do
        let cl = Winfo.class_name !f in
        if List.mem cl ["MMM"; "HFrame"] then raise Exit
        else f := Winfo.parent !f
      done;
      0
    with
      Exit ->
        truncate (float (Winfo.width !f) *. 0.95)
```

⟨*function* Table.text_align 409c⟩≡ (415)
```
  let text_align cell align =
    Text.tag_add cell "align" Frx_text.textBegin Frx_text.textEnd;
    Text.tag_configure cell "align"
      (match align with
        "right" -> [Justify Justify_Right]
      | "center" -> [Justify Justify_Center]
      | _ -> [Justify Justify_Left])
```

⟨*function* Table.dynamic_fight 409d⟩≡ (415)
```
  (* Fight for your life ! *)
  let dynamic_fight cell nowrap gameover align =
    match Winfo.class_name cell with
    | "Text" ->
        if !debug then
      Log.f (sprintf "DYNAMIC %s" (Widget.name cell));
        let when_finished () =
      if !debug then
        Log.f (sprintf "Switching %s to vertical resize"
                       (Widget.name cell));
          (* in all cases, we have to grow vertically *)
         let scroll, _check = Fit.vert cell in
         Text.configure cell [YScrollCommand scroll];
          (* A posteriori updates for embedded windows
         List.iter
        (fun embedded ->
          bind embedded [[], Configure]
            (BindSet([], (fun _ ->
             bind embedded [[], Configure] BindRemove;
             Frx_after.idle check;
```

```
                ())))))
            (Text.window_names cell)
           *)
        in
        let scroll, check = Fit.horiz cell gameover (
      let first_time = ref true in
      (fun () ->
        if !first_time then begin
          first_time := false;
          text_align cell align;
          if not nowrap then Text.configure cell [Wrap WrapWord];
          when_finished()
      end))
        in
        Text.configure cell [XScrollCommand scroll];
        check()
    | s ->
        if !debug then
      Log.f (sprintf "Table.dynamic_size: unknown children class %s" s);
        assert false
```

⟨*function* Table.fixed_size 410a⟩≡ (415)

```
  (* We know the size in pixels *)
  let fixed_size cell width nowrap align =
    match Winfo.class_name cell with
    | "Text" ->
        if !debug then
      Log.f (sprintf "FIXED %s to %d" (Widget.name cell) width);
        if not nowrap then Text.configure cell [Wrap WrapWord];
        Fit.fixed_horiz cell width;
        (* we have to do alignment here, because it kills horizontal resizing *)
        text_align cell align;
        (* in all cases, we have to grow vertically *)
        let scroll, check = Fit.vert cell in
        Text.configure cell [YScrollCommand scroll];
        (* A posteriori updates for embedded windows
        List.iter
      (fun embedded ->
        bind embedded [[], Configure]
          (BindSet([], (fun _ ->
            bind embedded [[], Configure] BindRemove;
            Frx_after.idle check;
               ())))))
          (Text.window_names cell);
         *)
        check()
    | s ->
        if !debug then
      Log.f (sprintf "Table.dynamic_size: unknown children class %s" s);
        assert false
```

⟨*function* Table.sizing 410b⟩≡ (415)

```
  (*
   * Determine how we should set resizing for our cells
   *  table.width contains the specified width for the table
   *  contextwidth was the width computed the context of the table
   *)
  let sizing table nowrap _width =
    (* For cells of given width and colspan 1, set a col minsize *)
    let colwidths = Array.make (Array.length table.slots) 0 in
    let setcolwidth col n =
      if n > colwidths.(col) then begin
        colwidths.(col) <- n;
        if !debug then
      Log.f (sprintf "%s col %d minsize %d"
                  (Widget.name table.master_widget) col n);
        Grid.column_configure table.master_widget col [Minsize (Pixels n)]
      end
    in
    (* second pass to see if we have a proper column width for these *)
    let dynamic = ref [] in
    let add_dynamic w f = dynamic := (w,f) :: !dynamic in
    (* adjust sizes with padding/spacing *)
    let adjust w = w - 2 * table.cellspacing - 2 * table.cellpadding in
    (* Set initial size and dynamic resizing *)
    List.iter (function w,(_,col,_,cspan,cellwidth,_,align) ->
      (* set initial width from images *)
      let _initw = Fit.set_initial_width w
      (* set initial height from line number *)
      and _ = Fit.set_initial_height w in
      match cellwidth with
        FixedWidth n ->
      if cspan=1 then setcolwidth col n; (* this col is at least n*)
      fixed_size w (adjust n) nowrap align
      | UnknownWidth bound ->
      add_dynamic w bound
      | _ -> assert false)
      (List.rev table.slaves);
    (* second pass on dynamics : if we know exactly the size of the
       cell because we know exactly the size of each column it belongs to
       then set it *)
    List.iter (fun (w, f) ->
      let unknown_col = ref false
      and width = ref 0 in
      let (_,col,_,cspan,_,_,align) = List.assoc w table.slaves in
      for i = col to col + cspan - 1 do
        width := !width + colwidths.(i);
        if colwidths.(i) = 0 then unknown_col := true
      done;
      if not !unknown_col then fixed_size w (adjust !width) nowrap align
      else dynamic_fight w nowrap f align)
      !dynamic
```

⟨*function* Table.packem 411a⟩≡ (415)

```
  (* TODO: alignment *)
  let packem table =
    let default_opts = [Sticky "nswe";
      PadX (Pixels table.cellspacing); PadY (Pixels table.cellspacing);
      IPadX (Pixels table.cellpadding); IPadY (Pixels table.cellpadding)]
    in
    List.iter
      (fun (w, (row,col,rspan,cspan, _, _, _)) ->
         (* Sticky opt gives Expand true, Fill Both *)
         grid [w] ([Row row; Column col; RowSpan rspan; ColumnSpan cspan]
                   @default_opts))
      table.slaves
```

⟨*function* Table.get_slot 411b⟩≡ (415)

```
  (*
   * Slots represent, by column, the number of "pending" row-spanning cells
   * If this number is zero, the slot is empty. When we allocate slots for
   * col-spanning cells, we keep these slots contiguous (case of overlapping
   * cells)
   *)

  let get_slot table needed_cols rspan =
    (* First free slot in cur_col *)
    let rec first_free n =
      if n < Array.length table.slots then
        if table.slots.(n) = 0 then n
        else first_free (n+1)
      else raise Not_found in
    try
      let first = first_free table.cur_col in
      (* Grow if overflow  (the next free would be first + needed_cols) *)
      if first + needed_cols > Array.length table.slots then
        table.slots <-
           Array.append table.slots
             (Array.make (first + needed_cols - (Array.length table.slots))
                           rspan);
      (* Mark used *)
      for i = first to first + needed_cols - 1 do
        table.slots.(i) <- max rspan table.slots.(i)
      done;
      table.cur_col <- first + needed_cols;
      first
    with
      Not_found -> (* Grow *)
        let first = Array.length table.slots in
        table.slots <- Array.append table.slots (Array.make needed_cols rspan);
        table.cur_col <- Array.length table.slots;
        first
```

⟨*function* Table.next_row 412a⟩≡ (415)

```
  let next_row table =
    for i = 0 to Array.length table.slots - 1 do
      table.slots.(i) <-
         match table.slots.(i) with
        0|1 -> 0
      | n -> n-1
    done
```

⟨*function* Table.create 412b⟩≡ (415)

```
  (*
   * The table manager
   * [top] is the frame that will be embedded in the text widget
   *)


  let create top tag contextwidth =
   let width =
     try length_of_string (get_attribute tag "width")
     with Not_found -> Nolength
   and cellpadding =
     try int_of_string (get_attribute tag "cellpadding")
     with Not_found | Failure "int_of_string" -> 0
   and cellspacing =
     try int_of_string (get_attribute tag "cellspacing")
     with Not_found | Failure "int_of_string" -> 0
   and bwidth =
     try int_of_string (get_attribute tag "border")
     with Not_found -> 0
        | Failure "int_of_string" -> 1
   and nowrap = has_attribute tag "nowrap"
   (* align attribute is ignored (flow of text) *)
   in
   Frame.configure top [BorderWidth (Pixels bwidth); Relief Raised];
   let tab = {
      master_widget = top;
      slaves = [];
      _width = width;
      cur_col = 0;
      cur_row = -1; (* Start with TR *)
      slots = [||];
      cols = [];
      cellpadding = cellpadding;
      cellspacing = cellspacing} in

   (* Compute (if possible) the width of this table *)
   (* Set up the resize condition for cells of this table *)
   let size, bound =
     match width with
       Nolength | LengthRel _ -> (* assume then 100% of context *)
         begin match contextwidth with
       TopWidth ->
         let w = topwidth tab.master_widget in
         None, Fit.bound_check tab.master_widget w
         | FixedWidth n -> (* size of parent cell *)
         Some n, Fit.bound_check tab.master_widget n
         | UnknownWidth f ->
             (* the previous bound must have been reached
          * and we (the frame) may occupy 100% of the context
          * (the text widget). Adjust to 95% for tuning.
          *)
         None, (fun () ->
           f() &&
             let w1 = Winfo.reqwidth top
             and w2 = Winfo.width (Winfo.parent top) in
             if !debug then
           Log.f (sprintf "Grow check %s=%d %s=%d"
                      (Widget.name top) w1
                      (Widget.name (Winfo.parent top)) w2);
              float w1 >= (float w2 *. 0.95))
         end
     | LengthPixels n ->
         Some n, Fit.bound_check tab.master_widget n
     | LengthRatio r -> (* check the context *)
         begin match contextwidth with
       TopWidth ->
         let w = truncate (float (topwidth tab.master_widget) *. r) in
         Some w, Fit.bound_check tab.master_widget w
         | FixedWidth n -> (* size of parent cell *)
         let w = truncate (float n *. r) in
         Some w, Fit.bound_check tab.master_widget w
         | UnknownWidth f ->
         None,
        (* the previous bound must have been reached,
           and we must occupy the ratio *)
         (fun () -> f()
             &&
           (let w1 = Winfo.reqwidth top in
            let w2 = Winfo.width (Winfo.parent top) in
             if !debug then
           Log.f (sprintf "Grow check %s=%d %s=%d"
                      (Widget.name top) w1
                      (Widget.name (Winfo.parent top)) w2);
            w1 >= truncate (r *. float w2)))
         end
    in
    (* SPECIAL FIX FOR THE PEOPLE WHO DON'T RESPECT THE DTD : we always make
       sure we are in a row *)
    let in_row = ref false in

     {table_master = top;
      bound = bound;
      close_table =
       (fun () ->
         packem tab;
         sizing tab nowrap size);
      add_col =  (fun tag ->
        let span =
         try int_of_string (get_attribute tag "span")
         with Not_found | Failure "int_of_string" -> 1 in
        let width =
         (* Specification of the columns width (only pixel size supported) *)
         try Some (int_of_string (get_attribute tag "width"))
         with Not_found | Failure "int_of_string" -> None
        in
        for _i = 1 to span do
      tab.cols <- width :: tab.cols
        done);

      open_row = (fun _t ->
      tab.cur_col <- 0;
      tab.cur_row <- 1 + tab.cur_row;
          in_row := true;
          next_row tab);

      close_row = (fun () -> in_row := false);

      new_cell = (fun ctype attrs w align ->
        (* SPECIAL FIX FOR THE PEOPLE WHO DON'T RESPECT THE DTD *)
        if not !in_row then begin
      tab.cur_col <- 0;
      tab.cur_row <- 1 + tab.cur_row;
          in_row := true;
          next_row tab
      end;
        let opts = match ctype with
       HeaderCell -> [Relief Groove]
         | DataCell -> [Relief Sunken]
        in
        begin match Winfo.class_name w with
        | "Text" -> Text.configure w opts
        | _ -> assert false
        end;
        (* Tk needs spans > 0 *)
        let rspan =
      try max 1 (int_of_string (get_attribute attrs "rowspan"))
      with Not_found | Failure "int_of_string" -> 1
```

```
        and cspan =
      try max 1 (int_of_string (get_attribute attrs "colspan"))
      with Not_found | Failure "int_of_string" -> 1
        and width =
      try length_of_string (get_attribute attrs "width")
      with Not_found -> Nolength
        and height =
      try length_of_string (get_attribute attrs "height")
      with Not_found -> Nolength
        in
        (* find its place *)
        let real_col = get_slot tab cspan rspan in
        if !debug then
            Log.f (sprintf "Cell %s at row=%d col=%d rspan=%d cspan=%d"
                   (Widget.name w)
                       tab.cur_row real_col rspan cspan);
        (* compute the size of this cell, so that tables in it have
       something to work on *)
        let wconstraint = match width with
         Nolength | LengthRel _ -> UnknownWidth bound
        | LengthPixels n -> FixedWidth n
        | LengthRatio r ->
        if !strict_32 then UnknownWidth bound
        else
          (* variable size : do we know the size of the table ? *)
        match size with
          None -> UnknownWidth (fun () ->
            bound() &&
             let w1 = Winfo.reqwidth w
             and w2 = Winfo.width top in
             w1 >= truncate (float w2 *. r))
        | Some n -> FixedWidth (truncate (float n *. r))
        in
         (* We delay the gridding until we have all cells *)
         tab.slaves <-
            (w, (tab.cur_row, real_col,
             rspan, cspan,
             wconstraint, height, align))
           :: tab.slaves;
         wconstraint
         )}
```

⟨display/table.ml 415⟩≡

```
  open Printf
  open Tk
  open Html
  open Htmlfmt
```

  ⟨*constant* Table.debug 408b⟩
  ⟨*constant* Table.strict_32 408c⟩

  ⟨*type* Table.cell_type *(*display/table.ml*)* 408d⟩
  ⟨*type* Table.t *(*display/table.ml*)* 408e⟩

  ⟨*type* Table.table 409a⟩

  ⟨*function* Table.topwidth 409b⟩

  ⟨*function* Table.text_align 409c⟩

⟨*function* Table.dynamic_fight 409d⟩

⟨*function* Table.fixed_size 410a⟩

⟨*function* Table.sizing 410b⟩

⟨*function* Table.packem 411a⟩

⟨*function* Table.get_slot 411b⟩

⟨*function* Table.next_row 412a⟩

⟨*function* Table.create 412b⟩

F.8.31 display/htframe.ml

⟨*constant* Htframe.geom_sep 416a⟩≡ (421a)
```
  (* geometry specs *)
  let geom_sep = Str.regexp "[ \t\n]+\\|\\([ \t\n]*,[ \t\n]*\\)"
```

⟨*function* Htframe.parse_geom 416b⟩≡ (421a)
```
  let parse_geom s = List.map Html.length_of_string (Str.split geom_sep s)
```

⟨*function* Htframe.figure_geom 416c⟩≡ (421a)
```
  (* to deal with relative length n*, we have to combine relD and absD
   *   n* is n fragments of Total - Fixed
   *   = (Total - Fixed) * n/Sigma_n
   *   = Total * n/Sigma_n - Fixed * n/Sigma_n
   *   -> -relD (n/Sigma_n) -absD (-Fixed * n/Sigma_n)
   *)
  let figure_geom l =
    (* compute the amount of fixed size *)
    let fixed = ref 0
    and totalrel = ref 0 in
    List.iter (function
        LengthPixels n -> fixed := n + !fixed
      | LengthRel n -> totalrel := n + !totalrel
      | _ -> ())
      l;
    if !totalrel = 0 then (* simple case *)
      List.map (fun x -> [x]) l
    else
      List.map (function
        | LengthRel n ->
        let ratio = float n /. float !totalrel in
        let opts = [LengthRatio (min ratio 1.)] in
        if !fixed = 0 then opts
        else
          opts @ [LengthPixels (- (truncate (float !fixed *. ratio)))]
        | x -> [x])
        l
```

⟨*type* Htframe.frame 417a⟩≡ (421a)
```
  type frame = {
      frame_src : string;
      frame_name : string;
      frame_scrolling : string; (* yes | no | auto *)
      frame_opts : Tk.options list;
      frame_params : (string * string) list;
    }
```

⟨*type* Htframe.frameset 417b⟩≡ (421a)
```
  and frameset =
      int ref * int ref * celldesc array array
```

⟨*type* Htframe.cell_contents 417c⟩≡ (421a)
```
  and cell_contents =
    | Frame of frame
    | Frameset of frameset
```

⟨*type* Htframe.celldesc 417d⟩≡ (421a)
```
  and celldesc = {
      cell_width : Html.length list;
      cell_height : Html.length list;
      mutable cell_contents : cell_contents option
    }
```

⟨*function* Htframe.ignore_fo 417e⟩≡ (421a)
```
  (* This is morally for the <noframes> section *)
  let ignore_fo f = {
    new_paragraph = (fun () -> ());
    close_paragraph = (fun () -> ());
    print_newline = (fun _b -> ());
    print_verbatim = (fun _s -> ());
    format_string = (fun _s -> ());
    hr = (fun _l _n _b -> ());
    bullet = (fun _n -> ());
    set_defaults = (fun _s _l -> ());
    push_attr = (fun _l -> ());
    pop_attr = (fun _l -> ());
    isindex = (fun _s _s' -> ());
    start_anchor = (fun () -> ());
    end_anchor = (fun _h -> ());
    add_mark = (fun _ -> ());
    create_embedded = (fun _a _w _h -> Frame.create f []);
    see_frag = (fun _ -> ());
    flush = (fun () -> ());
    }
```

⟨*constant* Htframe.ignore_close 417f⟩≡ (421a)
```
  let ignore_close = fun _ -> ()
```

⟨*function* Htframe.add_frames 417g⟩≡ (421a)
```
  let add_frames load_frames kill_body top mach =
    (* we start from an initial cell of "full size" *)
    let initial_cell = {
      cell_width = [LengthRatio 100.];
      cell_height = [LengthRatio 100.];
      cell_contents = None
    } in
    (* framesets can be defined recursively, this is our context stack *)
    let framesets = ref ([] : frameset list) in
```

```
(* each nested frame/frameset occupies a cell in its parent
 * the current cell to occupy is defined by ri/rj.
 *)
let next_cell set ri rj =
  incr rj;
  if !rj >= Array.length set.(!ri) then begin
    rj := 0; incr ri
  end
in

(* Create the frames with the proper placing, launch the display *)
let doit () =
  (* compute the real frames (the ones with embedded documents) *)
  let frames = ref ([] : (frame * Widget.widget) list) in
  let framesym = Mstring.egensym "framecell" in
  (* in some top window, place the given cell and proceed with its
   * contents recursively. [pos] defines the placing options in x/y
   *)
  let rec docell top cell pos =
    let f =
  Frame.create_named top
    (if cell == initial_cell then "frames"
     else framesym()) [Class "HFrame"] in
    let place_opts = ref (In top :: pos) in
    (* the displacement caused by this cell in its parent *)
    let delta_x = ref 0 and delta_relx = ref 0.
    and delta_y = ref 0 and delta_rely = ref 0.
    in
    List.iter (function
     | Nolength | LengthRel _ -> assert false
     | LengthPixels n ->
      place_opts := Width (Pixels n) :: !place_opts;
      delta_x := n
     | LengthRatio w ->
      place_opts := RelWidth w :: !place_opts;
      delta_relx := w)
  cell.cell_width;
    List.iter (function
     | Nolength | LengthRel _ -> assert false
     | LengthPixels n ->
      place_opts := Height (Pixels n) :: !place_opts;
      delta_y := n
     | LengthRatio w ->
      place_opts := RelHeight w :: !place_opts;
      delta_rely := w)
  cell.cell_height;
    (* place the cell, unless it is the top cell *)
    if cell == initial_cell then begin
  pack [f] [Expand true; Fill Fill_Both];
  Pack.propagate_set f false
    end else place f !place_opts;
    (* proceed with its contents *)
    begin match cell.cell_contents with
    | None -> () (* this is an error ! *)
    | Some (Frame frame) ->
    (* just store it so we can run the viewers later.
       (we need to have all frames in order to give proper navigation
       context for links with targets) *)
    frames := (frame, f) :: !frames;
    Frame.configure f frame.frame_opts
    | Some (Frameset (_,_,rows)) ->
    (* this is again a frameset. Thus [f] is still only a container *)
    (* positions of the embedded cells *)
    let curabs_x = ref 0 and curabs_y = ref 0
    and currel_x = ref 0. and currel_y = ref 0. in
    (* placer options for each embedded cell *)
    let curpos () = [X (Pixels !curabs_x); RelX !currel_x;
             Y (Pixels !curabs_y); RelY !currel_y] in
    (* Iterate on cells *)
    Array.iter (fun row ->
      (* for each row, we start horizontally at 0 *)
      curabs_x := 0; currel_x := 0.;
      (* the vertical size contributed by this row. It's constant
         for all cells in the row, but we compute it n times...*)
      let row_delta_y = ref 0
      and row_delta_rely = ref 0. in
      (* now iterate on each cell in this row (eg on columns) *)
      Array.iter (fun cell ->
        (* place this cell and return its occupation *)
        let delta_x, delta_relx, delta_y, delta_rely =
      docell f cell (curpos()) in
        (* switch current horiz position, store vert occupation *)
        curabs_x := delta_x + !curabs_x;
        currel_x := delta_relx +. !currel_x;
        row_delta_y := delta_y;
        row_delta_rely := delta_rely
        ) row;
      (* we finished the row. Move vertically now *)
      curabs_y := !curabs_y + !row_delta_y;
      currel_y := !currel_y +. !row_delta_rely)
      rows
    end;
    (* our caller expects us to return our size *)
    !delta_x, !delta_relx, !delta_y, !delta_rely
  in
  (* The initial cell is always at 0/0 *)
  ignore (docell top initial_cell [X (Pixels 0); Y (Pixels 0)]);
  (* And now proceed with frame loading *)
  load_frames !frames;
  (* some people put a body outside the noframes section, so we should
     ignore it completely if we saw frames. And we must kill the body
     if it was already created *)
  kill_body();
  mach#add_tag "body"
    (fun _fo _t -> mach#look_for EOF) ignore_close

in
mach#add_tag "frameset"
    (fun _fo t ->
  let rows = get_attribute t "rows"
  and cols = get_attribute t "cols" in
  let newset =
    ref 0, ref 0,
    Array.of_list
      (List.map (fun h ->
        Array.of_list (List.map  (fun w ->
      { cell_width = w;
        cell_height = h;
        cell_contents = None})
```

```
                (figure_geom (parse_geom cols))))
        (figure_geom (parse_geom rows)))
  in
  match !framesets with
  | [] ->
      (* if there two or more non-nested framesets, we will cause
         an error later *)
      if initial_cell.cell_contents <> None then
        raise (Invalid_Html "illegal <frameset>")
      else begin
        initial_cell.cell_contents <- Some (Frameset newset);
        framesets := newset :: !framesets
      end
  | (ri, rj, set)::_l ->
      if !ri >= Array.length set then
        raise (Invalid_Html "no room for <frameset> in this <frameset>")
      else begin
        set.(!ri).(!rj).cell_contents <-  Some (Frameset newset);
        next_cell set ri rj;
        framesets := newset :: !framesets
      end)
    (fun _t ->
  match !framesets with
  | [] ->
      raise (Invalid_Html "unmatched </frameset>")
  | [_x] -> (* the last one *)
      framesets := [];
      doit()
  | _x::l ->
      framesets := l);

mach#add_tag "frame"
  (fun _fo t ->
    match !framesets with
    | [] -> raise (Invalid_Html "<frame> outside <frameset>")
    | (ri, rj, set) :: _ ->
    if !ri >= Array.length set then
      raise (Invalid_Html "no room for <frame> in this <frameset>")
    else begin
      try
        let src = get_attribute t "src"
        and name = try get_attribute t "name" with Not_found -> ""
        and border =
         try int_of_string (get_attribute t "frameborder")
         with Failure "int_of_string" ->
      (* compatibility ? *)
        if String.lowercase_ascii (get_attribute t "frameborder") = "no"
        then 0 else 1
        and scrolling = String.lowercase_ascii (get_attribute t "scrolling")
        in
        let borderopts =
         if border = 0 then [BorderWidth (Pixels 0)]
         else [BorderWidth (Pixels border); Relief Ridge]
        in
        set.(!ri).(!rj).cell_contents <-
       Some (Frame { frame_src = src;
                  frame_name = name;
                  frame_opts = borderopts;
                  frame_scrolling = scrolling;
                  frame_params = t.attributes });
```

```
            next_cell set ri rj
          with
            Not_found ->
             raise (Invalid_Html "missing src in <FRAME>")
        end)
      ignore_close;

    (* note: <noframes> does not necessarily cover the whole body of the
     * document. It may only hide a toc which is displayed in another frame.
     * Basically, <noframes> doesn't imply there was a <frameset> in the
     * same document. Of course, we should interpret noframes ONLY if the
     * other frame supposed to contain the info IS displayed. But we don't
     * know that, do we ?
     *)
    mach#add_tag "noframes"
      (fun _fo _t ->
        mach#push_formatter (ignore_fo top);
        mach#look_for (CloseTag "noframes"))

      (fun _t -> mach#pop_formatter |> ignore; ())
```

⟨display/htframe.ml 421a⟩≡

```
  open Tk
  open Html
  open Htmlfmt

  (* Frames *)
```

  ⟨*constant* Htframe.geom_sep 416a⟩
  ⟨*function* Htframe.parse_geom 416b⟩

  ⟨*function* Htframe.figure_geom 416c⟩

```
  (* We build this data structure when parsing FRAMESET *)
```

  ⟨*type* Htframe.frame 417a⟩

  ⟨*type* Htframe.frameset 417b⟩

  ⟨*type* Htframe.cell_contents 417c⟩

  ⟨*type* Htframe.celldesc 417d⟩

  ⟨*function* Htframe.ignore_fo 417e⟩

  ⟨*constant* Htframe.ignore_close 417f⟩

  ⟨*function* Htframe.add_frames 417g⟩

F.8.32 display/textw_fo.mli

⟨*signature* Textw_fo.html_bg 421b⟩≡ (422b)

```
  val html_bg : string ref
```

⟨*signature* Textw_fo.html_fg 421c⟩≡ (422b)

```
  val html_fg : string ref
```

⟨*signature* Textw_fo.usecolors 421d⟩≡ (422b)

```
  val usecolors : bool ref
```

⟨*signature* Textw_fo.internal_buffer 422a⟩≡ (422b)
```
  val internal_buffer : int ref
```

⟨display/textw_fo.mli 422b⟩≡

```

  ⟨signature Textw_fo.html_bg 421b⟩
  ⟨signature Textw_fo.html_fg 421c⟩

  ⟨signature Textw_fo.usecolors 421d⟩
  ⟨signature Textw_fo.internal_buffer 422a⟩

  ⟨signature Textw_fo.create 129b⟩
```

F.8.33 display/textw_fo.ml

⟨*constant* Textw_fo.html_bg 422c⟩≡ (422g)
```
  (* Default background and foreground colors *)
  let html_bg = ref "white"
```

⟨*constant* Textw_fo.html_fg 422d⟩≡ (422g)
```
  let html_fg = ref "black"
```

⟨*constant* Textw_fo.usecolors 422e⟩≡ (422g)
```
  (* Preference settings *)
  let usecolors = ref true     (* use colors (fg/bg) specified in document *)
```

⟨*constant* Textw_fo.internal_buffer 422f⟩≡ (422g)
```
  let internal_buffer = ref 4000
```

⟨display/textw_fo.ml 422g⟩≡
```
  open Printf
  open Tk
  open Frx_text
  open Hyper


  open Htmlfmt
  open Fonts


  (* Text widget formatter for the HTML Display Machine
   * The main function builds a GfxHTML, in two cases
   *    1- normal (viewing an HTML document)
   *    2- nested (a cell in a table)
   *    3-
   *)

  ⟨constant Textw_fo.html_bg 422c⟩
  ⟨constant Textw_fo.html_fg 422d⟩

  ⟨constant Textw_fo.usecolors 422e⟩
  ⟨constant Textw_fo.internal_buffer 422f⟩

  ⟨function Textw_fo.create 129c⟩
```

F.8.34 display/html_disp.mli

```
⟨display/html_disp.mli 423a⟩≡

  ⟨signature Html_disp.verbose 246g⟩
  ⟨signature Html_disp.attempt_tables 152c⟩

  ⟨signature class Html_disp.imgloader 171c⟩

  ⟨signature class Html_disp.machine 27a⟩

  ⟨signature Html_disp.add_hook 188a⟩

  ⟨signature functor Html_disp.Make 123h⟩
```

F.8.35 display/html_disp.ml

```
⟨display/html_disp.ml 423b⟩≡
  (* HTML Display Machine *)

  ⟨constant Html_disp.verbose 246h⟩
  ⟨constant Html_disp.attempt_tables 152d⟩

  ⟨function Html_disp.lowernumber 147a⟩
  ⟨function Html_disp.uppernumber 147b⟩

  ⟨constant Html_disp.romans 147c⟩
  ⟨function Html_disp.roman 147d⟩

  ⟨class Html_disp.imgloader 172b⟩

  ⟨class Html_disp.machine 28⟩

  ⟨constant Html_disp.user_hooks 188c⟩
  ⟨function Html_disp.add_hook 188b⟩

  ⟨constant Html_disp.default_fo 26f⟩

  ⟨function Html_disp.push_style 139a⟩
  ⟨function Html_disp.pop_style 139b⟩

  ⟨functor Html_disp.Make 125a⟩
```

F.9 gui/

F.9.1 gui/about.mli

```
⟨signature About.create_tachy 423c⟩≡                                    (423d)
  val create_tachy : Widget.widget -> Low.tachymeter

⟨gui/about.mli 423d⟩≡
  ⟨signature About.create_tachy 423c⟩

  ⟨signature About.f 44c⟩
```

F.9.2 gui/about.ml

⟨*constant* About.tachy_data 424a⟩≡ (425b)

```
(* inside bitmap, circle is in +16+7 +66+57, radius 25 *)

let tachy_data = "GIF\056\057aP\000A\000\227\000\000\000\000\000\
\044\044\044\060\000\000YYY\138\138\138\154\154\154\170\
\170\170\186\186\186\203\203\203\219\219\219\231qq\235\235\
\235\243yy\255\255\255\000\000\000\000\000\000\033\249\004\
\001\000\000\009\000\044\000\000\000\000P\000A\000\000\004\
\255\048\201I\171\189\024H\157\056\199\096\040\142\164\213\
Hg\146\166e\235\190p\060\013\052\045\223DN\000\
M\225\223\148\001\133\005\020\233\142\187\158oY\184\009\
\039\196\162\005I\229\049\153\006\131E\032z\162\164\023\
\042\178\144\188\250\178\232J\180\226U\021\063\146\129\248\
\136\174c\235\233\009\220\212\232\175a\041\052sx\132\
\133\133P\096\033p\130H\134\142\143\006\007z\137\032\
\129rt\144\153x\007\156\136\148\147\051\151\058\154\164\
Y\156\156\004\160\159n\009\003\000\152\165\164\167I\158\
\171\173r\175\004\177\187\007I\157\018\092\171\140\059\187\
\197\060\167\146\095\148\195\186\197\197\200\008\008\182\173\176\
\206\188\007\209\210\194\213\214\187\217\209\171\141\221\206\223\
\218\137\226\227\228\223\148\220\233\222\217\231\237\238\165\229\
\096\242\143\001\001\003\001\006\174\053\006\249\000\210\131\039\
\229\158\035\125\253\250\001\232\055\064\033\141\129\230\054\196\
\064G\010\192\190\044\251\030fA\056\240\194\031\035\163\
\255b\185J\008\208\031\198\128\177\216\133\044\165\049\203\
\194\135\015\045\166\180\176\167\132AC\045\255\213\216\185\
\203\227\011\138\243\172\197\203\017t\030\205\018\031n\022\
\037u\164\150\136\020J\151f\002\064\181f\139\170\011\
\221\037\152\231\039\133\213\011\031\186J\029\167\160\172\002\
\167\033\160\046\025\219\205\236Y\025w\216\022s\043\241\
\197\021\185\187\220\190e\229\034\046\094Mt\129\248\253\
\011\041p\135\024\131\009\059\050\204\183\197\093\197\143\220\
\050\144\178\022r\228\178\147\041\023\176\092X\001\131\204\
E\054s\094\236\249s\162\209\139\063\131\150\130\186P\
i\211\148Z\215y\189\026\140l\052\170a\127\186M\
\123Z\029N\144\049\171\158\182\053R\175\172x\001\044\
\016N\092\002\170cx\023\044P\094\182\249\004\095\192\
\199J\151\174\220\250\016d\217\231m\031\239\253\176s\
\240\008\196\143\151\190\225k\034\022\208\178u\091\191\125\
\210G\091\156\202\201\039E\159\252\144\251\211\232\183N\
\045qxH\208\095\125\229\149\032\096\005\180Lp\096\
\130\048\012HA\131\009\244\007a\056\060\024\232\223\133\
\024\158\192\030\135\032\134\040\226\136\036\194\016\001\000\059\
"
```

⟨*constant* About.park_data 424b⟩≡ (425b)

```
let park_data =
"#define break_width 15
#define break_height 11
static char break_bits[] = {
   0x0c, 0x18, 0xf4, 0x17, 0x3a, 0x2e, 0xba, 0x2d, 0xb9, 0x4d, 0x3d, 0x5e,
   0xb9, 0x4f, 0xba, 0x2f, 0xba, 0x2f, 0xf4, 0x17, 0x08, 0x08};
"
```

⟨*constant* About.pi 424c⟩≡ (425b)

```
let pi = 3.1415926
```

⟨*constant* About.log10 424d⟩≡ (425b)

```
let log10 = log 10.0
```

⟨*function* About.create_tachy 425a⟩≡ (425b)

```
  let create_tachy top =
    let o = new default_tachy top in
    o#start;
```

⟨gui/about.ml 425b⟩≡

```
  open Tk

  ⟨function About.f 44d⟩

  (* Tachymeter *)

  (* gif is 80x65 *)
  ⟨constant About.tachy_data 424a⟩
  ⟨constant About.park_data 424b⟩

  ⟨constant About.pi 424c⟩
  ⟨constant About.log10 424d⟩

  class default_tachy (top : Widget.widget) =
   object (self)
    (* val top = top *)
    val mutable canvas = top (* dummy initialisation *)
    val mutable alive = false

    (* Various components of the canvas, all with dummy init values *)
    val mutable i_park = Tag "none"
    val mutable kilos = Tag "none"
    val mutable aig = Tag "none"
    val mutable pendings = Tag "none"


    (* this one is private *)
    method start =
      let c =
        Canvas.create_named top "tachymeter"
         [Width (Pixels 80); Height (Pixels 80);
           BorderWidth (Pixels 0);
           HighlightThickness (Pixels 0);
           TakeFocus true (* pl3 fix *)] in
      (* Use colors so that images are not transparent *)
      let tachy_image =
        begin
         try
        let bgc = Tk.cget c CBackground in
             Protocol.tkCommand
          [|Protocol.TkToken "set";
            Protocol.TkToken "TRANSPARENT_GIF_COLOR";
            Protocol.TkToken bgc |]
         with _ -> ()
        end;
      (* Agghaaa !!! TCL/TK doesn't support -data for GIF !!! *)
        let file = Msys.mktemp "tachy.gif" in
        let oc = open_out_bin file in
        output_string oc tachy_data;
        close_out oc;
        let img = Imagephoto.create [File file] in
        Msys.rm file;
        img
      and _park_image =
        Imagebitmap.create [Data park_data; Foreground Red] in

      i_park <-
        Canvas.create_rectangle c
       (Pixels 72) (Pixels 3)
       (Pixels 75) (Pixels 6) [FillColor Black];

      kilos <-
        Canvas.create_text c (Pixels 40) (Pixels 73) [Text "0"];

      aig <-
        Canvas.create_line c [Pixels 41; Pixels 32; Pixels 41; Pixels 57]
                         [Width (Pixels 2)];
      pendings <-
        Canvas.create_text c (Pixels 70) (Pixels 60) [Text "0"];

      let i_tachy =
        Canvas.create_image c (Pixels 0) (Pixels 0)
         [ImagePhoto tachy_image; Anchor NW]

      in

      Canvas.lower_bot c pendings;
      (* All other items must be put above the background image *)
      List.iter (fun i -> Canvas.raise_above c i i_tachy)
        [kilos; aig; i_park];

      bind c (Glevents.get "tachy_about") (BindSet ([], (fun _ -> f ())));

      bind c (Glevents.get "tachy_gc") (BindSet ([], (fun _ -> Frx_mem.f())));

      bind c [[], Destroy] (BindSet ([], (fun _ -> alive <- false)));

      pack [c][];
      alive <- true;
      canvas <- c

    val mutable last_speed = 0.
    val mutable last_total = 0
    val mutable idle = false

    method update speed total =
      if speed = 0.0 then begin
        if not idle then begin
      Canvas.configure_rectangle canvas i_park [FillColor Black;
                            Outline Black];
      idle <- true
        end
      end
      else begin
        Canvas.configure_rectangle canvas i_park [FillColor Green;
                           Outline Green];
        idle <- false
      end;
      if total <> last_total then
        Canvas.configure_text canvas kilos [Text (string_of_int total)];
      last_total <- total;
      let speed = if speed = 0. then 0. else log speed in
         (* Smooth *)
      let speeds = (last_speed +. speed) /. 2. in
```

```
      if abs_float (speeds -. last_speed) > 0.1 then begin
        last_speed <- speeds;
        let v = speeds /. log10 in
        let angle = v /. 4.0 *. pi in
        let angle = if angle < 0.1 then 0.0 else angle in
        let x = 41.0 -. (sin angle *. 25.0)
        and y = 32.0 +. (cos angle *. 25.0) in
        Canvas.coords_set canvas aig
      [Pixels 41; Pixels 32;
        Pixels (truncate x); Pixels (truncate y)];
        Low.update_idletasks()
      end

    method report_cnx n =
      if Winfo.exists canvas then
        if n = 0 then begin
      Canvas.configure_text canvas pendings [Text ""];
         Canvas.lower_bot canvas pendings
        end
        else begin
      Canvas.configure_text canvas pendings
        [Text (string_of_int n)];
         Canvas.raise_top canvas pendings
        end

    method report_busy busy =
      if Winfo.exists canvas then
        if busy then begin
         Canvas.lower_bot canvas pendings;
      Canvas.configure_rectangle canvas i_park [FillColor Red;
                                 Outline Red];
      Low.update_idletasks()
        end
        else begin
         Canvas.raise_top canvas pendings;
      Canvas.configure_rectangle canvas i_park [FillColor Black;
                                 Outline Black]
        end

    method report_traffic tick_duration bytes_read sample_read =
      if alive then
        self#update (float sample_read *. 1000. /. float tick_duration)
      bytes_read

    method quit =
      alive <- false;
      destroy canvas

  end
```

⟨*function* About.create_tachy 425a⟩

```
    (o :> Low.tachymeter)
```

F.9.3 gui/fontprefs.ml

⟨*function* Fontprefs.fontspec2attrs 427⟩≡ (430)

```
  let fontspec2attrs s =
    let tokens = Mstring.split_str (fun c -> c='-') s in
    if List.length tokens <> 14 then
      failwith ("incomplete font specification: " ^ s)
    else (* should not fail *)
      let attrs = ref [] in
        (match List.nth tokens 1
         with "*" -> () | s -> attrs := (Family s) :: !attrs);
        (match List.nth tokens 2
         with "*" -> () | s -> attrs := (Weight s) :: !attrs);
        (match List.nth tokens 3
         with "*" -> () | s -> attrs := (Slant s) :: !attrs);
        (match List.nth tokens 6 with
           "*" -> ()
      | s -> try
               attrs := (FontIndex (font_index (int_of_string s))) :: !attrs
             with Failure "int_of_string" ->
           failwith ("pxlsz not an integer: " ^ s));
        !attrs
```

⟨*function* Fontprefs.attrs2fontspec 428a⟩≡ (430)

```
  let attrs2fontspec l =
    let rec family = function
       [] -> "*"
     | (Family s)::_ -> s
     | _x::l -> family l
    and weight = function
       [] -> "*"
     | (Weight s)::_ -> s
     | _x::l -> weight l
    and slant = function
       [] -> "*"
     | (Slant s)::_ -> s
     | _x::l -> slant l
    and pxlsz = function
       [] -> "*"
     | (FontIndex s)::_ -> string_of_int (Fonts.pxlsz s)
     | _x::l -> pxlsz l in

    sprintf "-*-%s-%s-%s-normal-*-%s-*-*-*-*-*-iso8859-1"
            (family l) (weight l) (slant l) (pxlsz l)
```

⟨*constant* Fontprefs.default_families 428b⟩≡ (430)

```
  (* Build a family menu *)
  let default_families =
    ["courier"; "helvetica"; "lucida"; "new century schoolbook";
     "times"; "fixed"; "*"]
```

⟨*function* Fontprefs.families 428c⟩≡ (430)

```
  let families () =
   Tkresource.stringlist "fontFamilies" default_families
```

⟨*function* Fontprefs.family_select 428d⟩≡ (430)

```
  let family_select top v =
    Optionmenu.create  top v (families())
```

⟨*constant* Fontprefs.default_weights 428e⟩≡ (430)

```
  (* Build a weight menu *)
  let default_weights = ["bold"; "medium"; "*"]
```

⟨*function* Fontprefs.weights 428f⟩≡ (430)

```
  let weights () =
    Tkresource.stringlist "fontWeights" default_weights
```

⟨*function* Fontprefs.weight_select 429a⟩≡ (430)
```
  let weight_select top v =
    Optionmenu.create top v (weights())
```

⟨*constant* Fontprefs.default_slants 429b⟩≡ (430)
```
  (* Build a slant menu *)
  let default_slants = ["r"; "i"; "o"; "*"]
```

⟨*function* Fontprefs.slants 429c⟩≡ (430)
```
  let slants () =
    Tkresource.stringlist "fontSlants" default_slants
```

⟨*function* Fontprefs.slant_select 429d⟩≡ (430)
```
  let slant_select top v =
    Optionmenu.create top v (slants())
```

⟨*function* Fontprefs.pixels 429e⟩≡ (430)
```
  (* Build a pixel size menu *)
  let pixels() =
    Tkresource.stringlist "fontPixels" Fonts.default_sizes
```

⟨*function* Fontprefs.pixels_select 429f⟩≡ (430)
```
  let pixels_select top v =
    Optionmenu.create top v (pixels())
```

⟨*function* Fontprefs.font_select 429g⟩≡ (430)
```
  (* fontspecv is the variable used for the full X font name; it is used
   * internally (and for saving the prefs), and must be maintained consistent
   * with the displayed state.
   *   - initialisation time :
   *      given the attributes, write the X name in the variable
   *   - edition time:
   *      electric update of the variable and the styles
   *)

  let font_select top getattrs setattrs =
    let familyv = Textvariable.create_temporary top
    and weightv = Textvariable.create_temporary top
    and slantv = Textvariable.create_temporary top
    and pixelsv = Textvariable.create_temporary top
    and fontspecv = Textvariable.create_temporary top
    in
    let f = Frame.create top [] in
    let buttons =
      List.map2 (fun create v ->
                   Textvariable.set v "*";
                   let x,_ = create f v in x)
                [family_select; weight_select; slant_select; pixels_select]
                [familyv; weightv; slantv; pixelsv] in
    pack buttons [Side Side_Left];
    (* electric updates
     * Whenever one of the attributes changes, we must change the fontspec
     * and possibly recompute the attributes
     *)
    let setv _ =
      let font = sprintf "-*-%s-%s-%s-normal-*-%s-*-*-*-*-*-iso8859-1"
          (Textvariable.get familyv)
      (Textvariable.get weightv)
      (Textvariable.get slantv)
          (Textvariable.get pixelsv)
      in
      let attrs = fontspec2attrs font in
      Textvariable.set fontspecv font;
      setattrs attrs
    in
    List.iter (fun v ->
      let rec el () = Textvariable.handle v (fun () -> setv(); el()) in el())
      [familyv; weightv; slantv; pixelsv];

    (* initialisation from memory (v=fontspecv) *)
    let init_pref _v =
      let attrs = getattrs() in
      (* Set all variables; electric update does the rest *)
      List.iter (function
          Family s -> Textvariable.set familyv s
        | Weight s -> Textvariable.set weightv s
        | Slant s -> Textvariable.set slantv s
        | FontIndex s ->
            Textvariable.set pixelsv (string_of_int (Fonts.pxlsz s))
        | _ -> assert false)
        attrs
    (* initialisation from loaded strings (v=fontspecv) *)
    and set_pref v =
      let font = Textvariable.get v in
      let attrs = fontspec2attrs font in
      (* Set all variables; electric update rewrites everything (duh) *)
      List.iter (function
          Family s -> Textvariable.set familyv s
        | Weight s -> Textvariable.set weightv s
        | Slant s -> Textvariable.set slantv s
        | FontIndex s ->
            Textvariable.set pixelsv (string_of_int (Fonts.pxlsz s))
        | _ -> assert false)
        attrs
    in
    f, fontspecv, init_pref, set_pref
```

⟨gui/fontprefs.ml 430⟩≡

```
  open Printf
  open Tk
  (* Specify set of attributes of a font *)
  (* family, weight, slant, pxlsz *)
  (* We use a font string, and select only the relevant components *)
  (*
  -fndry-fmly-wght-slant-sWdth-adstyl-pxlsz-ptSz-resx-resy-spc-avgWdth-reg-enc
     0    1    2    3     4      5     6     7    8    9   10    11     12  13
  *)
  open Fonts
```

⟨*function* Fontprefs.fontspec2attrs 427⟩

⟨*function* Fontprefs.attrs2fontspec 428a⟩

⟨*constant* Fontprefs.default_families 428b⟩
⟨*function* Fontprefs.families 428c⟩

⟨*function* Fontprefs.family_select 428d⟩

⟨*constant* Fontprefs.default_weights 428e⟩
⟨*function* Fontprefs.weights 428f⟩

```
  ⟨function Fontprefs.weight_select 429a⟩

  ⟨constant Fontprefs.default_slants 429b⟩
  ⟨function Fontprefs.slants 429c⟩
  ⟨function Fontprefs.slant_select 429d⟩

  ⟨function Fontprefs.pixels 429e⟩
  ⟨function Fontprefs.pixels_select 429f⟩


  ⟨function Fontprefs.font_select 429g⟩
```

F.9.4 gui/gcache.mli

```
⟨signature Gcache.debug 431a⟩≡                                        (431i)
  val debug : bool ref

⟨signature Gcache.max_keep 431b⟩≡                                     (431i)
  val max_keep : int ref

⟨signature Gcache.kill 431c⟩≡                                         (431i)
  val kill : int -> unit
    (* [kill hkey] destroy all widget cached for navigator [hkey]
       If in history mode, accordingly remove from Cache documents
       that are not shared with other navigator windows
     *)

⟨signature Gcache.find 431d⟩≡                                         (431i)
  val find : int -> Document.id -> Viewers.display_info

⟨signature Gcache.add 431e⟩≡                                          (431i)
  val add : int -> Document.id -> Viewers.display_info -> unit

⟨signature Gcache.remove 431f⟩≡                                       (431i)
  val remove : int -> Document.id -> unit

⟨signature Gcache.displace 431g⟩≡                                     (431i)
  val displace : int -> Document.id -> unit

⟨signature Gcache.postmortem 431h⟩≡                                   (431i)
  val postmortem : unit -> unit

⟨gui/gcache.mli 431i⟩≡
  (* Cache by "widget unmapping"
   * For each navigator, we keep the list of displayed documents
   *)

  ⟨signature Gcache.debug 431a⟩

  ⟨signature Gcache.max_keep 431b⟩

  ⟨signature Gcache.kill 431c⟩

  ⟨signature Gcache.find 431d⟩
  ⟨signature Gcache.add 431e⟩
  ⟨signature Gcache.remove 431f⟩
  ⟨signature Gcache.displace 431g⟩

  ⟨signature Gcache.postmortem 431h⟩
```

F.9.5 gui/gcache.ml

⟨*constant* Gcache.debug 432a⟩≡ (434a)
```
  (* Cache by "widget unmapping"
   *  For each navigator, we keep the list of displayed documents
   *)

  let debug = ref false
```

⟨*constant* Gcache.max_keep 432b⟩≡ (434a)
```
  let max_keep = ref 5
    (* maximum number of cached widget in a given window *)
```

⟨*constant* Gcache.table 432c⟩≡ (434a)
```
  let table = (Hashtbl.create 37 :
                (int, (Document.id * display_info) list ref) Hashtbl.t)
```

⟨*function* Gcache.get_nav 432d⟩≡ (434a)
```
  let get_nav hkey =
    try
      Hashtbl.find table hkey
    with
      Not_found ->
        let r = ref [] in
         Hashtbl.add table hkey r;
         r
```

⟨*function* Gcache.find 432e⟩≡ (434a)
```
  (* Find a document in a given window
   * Called by the navigator when attempting to display a new request.
   * Also called by back/forward navigation in the history
   *)
  let find hkey did =
    let r = get_nav hkey in
    List.assoc did !r
```

⟨*function* Gcache.nocache 432f⟩≡ (434a)
```
  (* History mode: when we remove a document from the gcache, and that it
     was its only displayed instance, then we must also remove it from cache
   *)
  let nocache did =
   if !debug then Log.f
      (sprintf "Removing %s from cache" (Url.string_of did.document_url));
   let shared = ref false in
    Hashtbl.iter (fun _key dis -> if List.mem_assoc did !dis then shared := true)
      table;
   if not !shared then Cache.kill did
   else
     if !debug then Log.f "Don't, it's shared"
```

⟨*function* Gcache.displace 432g⟩≡ (434a)
```
  (* Removing only to redisplay (update) *)
  let displace hkey did =
    if !debug then Log.f
      (sprintf "Displacing %s in window %d" (Url.string_of did.document_url) hkey);
    try
      let r = Hashtbl.find table hkey in
      let di = List.assoc did !r in
       di#di_abort;
       di#di_destroy;
       r := Mlist.except_assoc did !r;
    with
       Not_found -> Log.debug "Gcache.remove failed !"
```

⟨*function* Gcache.add 433a⟩≡ (434a)
```
  (* Add a new display_info for a document in the cache *)
  let add hkey did di =
    try
      let r = Hashtbl.find table hkey in
       r := (did, di) :: !r;
      (* the problem is to find the correct ones to delete, because we are
         not sure that the older are really the older in history. Well.
       *)
      if List.length !r > !max_keep then
        let l = List.sort (fun (_,di) (_,di') -> di_compare di di') !r in
        let fluff = Mlist.tln l !max_keep in
      List.iter (fun (did,_) -> remove hkey did) fluff
    with
      Not_found -> ()
```

⟨*function* Gcache.kill 433b⟩≡ (434a)
```
  (* A window is being destroyed: kill all visible instances
   *  Note: there could be a document still being retrieved and displayed,
   *  but not present in the history.
   *)
  let kill hkey =
   if !debug then Log.f (sprintf "Killing gcache for nav %d" hkey);
   let r = get_nav hkey in
    List.iter (fun (_did, di) -> di#di_abort) !r;
    if !Cache.history_mode then begin
       let fluff = !r in
         r := []; (* so that we don't find them again *)
         List.iter (fun (did, _) -> nocache did) fluff
    end;
   Hashtbl.remove table hkey
```

⟨*function* Gcache.postmortem 433c⟩≡ (434a)
```
  let postmortem () =
    Hashtbl.iter (fun key dis ->
        Log.f (sprintf "Navigator %d" key);
        List.iter (fun (did,_) ->
       Log.f (sprintf "%s(%d)"
            (Url.string_of did.document_url)
            did.document_stamp))
          !dis)
     table
```

⟨*function* Gcache.sorry 433d⟩≡ (434a)
```
  (* If the normal cache gets full, we might *have* to destroy documents
   * that are visible. In that case, kill the gcache as well, so that
   * we don't get strange phenomenons such as image disappearing, ...
   *)

  let sorry did =
    Hashtbl.iter (fun key dis ->
      if List.mem_assoc did !dis then remove key did) table
```

⟨*toplevel* Gcache._1 433e⟩≡ (434a)
```
  let _ =
    Cache.cutlinks := sorry :: !Cache.cutlinks
```

```
⟨gui/gcache.ml 434a⟩≡
  open Printf
  open Document
  open Viewers

  ⟨constant Gcache.debug 432a⟩

  ⟨constant Gcache.max_keep 432b⟩

  ⟨constant Gcache.table 432c⟩

  ⟨function Gcache.get_nav 432d⟩

  ⟨function Gcache.find 432e⟩

  ⟨function Gcache.nocache 432f⟩

  ⟨function Gcache.remove 237i⟩

  ⟨function Gcache.displace 432g⟩

  ⟨function Gcache.add 433a⟩


  ⟨function Gcache.kill 433b⟩


  ⟨function Gcache.postmortem 433c⟩


  ⟨function Gcache.sorry 433d⟩


  ⟨toplevel Gcache._1 433e⟩
```

F.9.6 gui/plink.mli

```
⟨gui/plink.mli 434b⟩≡
  ⟨signature Plink.make 36f⟩
```

F.9.7 gui/plink.ml

```
⟨function Plink.dial 434c⟩≡                                                    (435b)
  let dial hlink err =
    let t = Toplevel.create Widget.default_toplevel [Class "Dialog"] in
    Focus.set t;
    Wm.title_set t (s_ "Malformed link error");

    let vuri = Textvariable.create_temporary t
    and vcontext = Textvariable.create_temporary t in

    Textvariable.set vuri hlink.h_uri;
    (match hlink.h_context with
      Some s -> Textvariable.set vcontext s
    | None -> ());

    let msg = match err with
        LinkResolve s -> s
```

```
     |  UrlLexing (s,_) -> s in

    let tit = Label.create t [Text (s_ "Malformed link error")]
    and fc,_ec = Frx_entry.new_labelm_entry t "Context" vcontext
    and fu,eu = Frx_entry.new_labelm_entry t "Relative" vuri
    and lmsg = Label.create t [Text msg]
    in
    let cancelled = ref false in
    let fb = Frame.create t [] in
      let bok = Button.create fb
              [Text "Ok"; Command (fun _ -> Grab.release t; destroy t)]
      and bcancel = Button.create fb
              [Text "Cancel"; Command (fun _ -> cancelled := true;
                                             Grab.release t; destroy t)]
      in

      pack [bok] [Side Side_Left; Expand true];
      pack [bcancel] [Side Side_Right; Expand true];
      pack [tit;fc;fu;lmsg;fb] [Fill Fill_X];
      Tkwait.visibility t;
      Focus.set eu;
      Grab.set t;
      Tkwait.window t;
      (* because the window gets destroyed, the variables too. *)
      if !cancelled then None
      else Some
           {h_uri = Textvariable.get vuri;
        h_context = (match Textvariable.get vcontext with
                      "" -> None
                     | s -> Some s);
            h_method = hlink.h_method;
        h_params = hlink.h_params}
```

⟨*function* Plink.make 435a⟩≡ (435b)

```
  (* Utility for catching link resolving errors *)
  let rec make hlink =
    try
      Www.make hlink
    with
      Invalid_link msg ->
        match dial hlink msg with
        | None -> raise (Invalid_link msg)
        | Some hlink -> make hlink
```

⟨gui/plink.ml 435b⟩≡

```
  open I18n
  open Tk
  open Hyper
```

  ⟨*function* Plink.dial 434c⟩

  ⟨*function* Plink.make 435a⟩

F.9.8 gui/prefs.mli

⟨*type* Prefs.pref_type 435c⟩≡ (437f)

```
  (* Exported so that we can plug applet preferences *)
  type pref_type =
   | Bool of bool ref
   | String of string ref
   | Int of int ref
   | Float of float ref
   | AbstractType of (Textvariable.textVariable -> unit) *
                     (Textvariable.textVariable -> unit)
                     (* init, set *)
```

⟨*type* Prefs.pref 436a⟩≡ (437f)

```
type pref = {
  packed_widget : Widget.widget;
  pref_variable : Textvariable.textVariable;
  pref_type : pref_type;
  pref_name : string; (* shall not contain : *)
  resource_name : string (* shall not contain : *)
}
```

⟨*type* Prefs.pref_family 436b⟩≡ (437f)

```
(* A family of preferences *)
type pref_family =
  {family_widget: Widget.widget;
   family_init : unit -> unit;
   family_save : unit -> string PrefMap.t;
   family_load : unit -> unit;
   family_title : string
  }
```

⟨*signature* Prefs.bool_pref 436c⟩≡ (437f)

```
val bool_pref : string -> bool ref -> Widget.widget -> pref
```

⟨*signature* Prefs.int_pref 436d⟩≡ (437f)

```
val int_pref : string -> int ref -> Widget.widget -> pref
```

⟨*signature* Prefs.float_pref 436e⟩≡ (437f)

```
val float_pref : string -> float ref -> Widget.widget -> pref
```

⟨*signature* Prefs.string_pref 436f⟩≡ (437f)

```
val string_pref : string -> string ref -> Widget.widget -> pref
    (* [<type>_pref name internal_location top] *)
```

⟨*signature* Prefs.option_pref 436g⟩≡ (437f)

```
val option_pref :
    string ->
    (Textvariable.textVariable -> unit) *
    (Textvariable.textVariable -> unit) * string list ->
    Widget.widget -> pref
```

⟨*signature* Prefs.abstract_bool_pref 436h⟩≡ (437f)

```
val abstract_bool_pref :
    string ->
      (Textvariable.textVariable -> unit) ->
      (Textvariable.textVariable -> unit) -> Widget.widget -> pref
```

⟨*signature* Prefs.abstract_string_pref 436i⟩≡ (437f)

```
val abstract_string_pref :
    string ->
      (Textvariable.textVariable -> unit) ->
      (Textvariable.textVariable -> unit) -> Widget.widget -> pref
```

⟨*signature* `Prefs.option_handlers` 437a⟩≡ (437f)
```
  val option_handlers :
      ('a * string) list ->
      (unit -> 'a) ->
      ('a -> unit) ->
      (Textvariable.textVariable -> unit) * (Textvariable.textVariable -> unit) *
      string list
```

⟨*signature* `Prefs.family` 437b⟩≡ (437f)
```
  val family :
      Widget.widget -> string -> (Widget.widget -> pref) list -> pref_family
```

⟨*signature* `Prefs.pref_error` 437c⟩≡ (437f)
```
  val pref_error : string -> unit
```

⟨*signature* `Prefs.resource_name` 437d⟩≡ (437f)
```
  val resource_name : string -> string
```

⟨*signature* `Prefs.define` 437e⟩≡ (437f)
```
  val define :
      Fpath.t ->
      (Widget.widget -> pref_family) list -> (unit -> unit) list -> unit -> unit
      (* [define filename pref_builders pref_mute]
         returns a function that displays the preference panel
       *)
```

⟨`gui/prefs.mli` 437f⟩≡
  ⟨*type* `Prefs.pref_type` 435c⟩

  ⟨*type* `Prefs.pref` 436a⟩

```
  module PrefMap : Map.S with type key = string
```

  ⟨*type* `Prefs.pref_family` 436b⟩

  ⟨*signature* `Prefs.bool_pref` 436c⟩
  ⟨*signature* `Prefs.int_pref` 436d⟩
  ⟨*signature* `Prefs.float_pref` 436e⟩
  ⟨*signature* `Prefs.string_pref` 436f⟩

  ⟨*signature* `Prefs.option_pref` 436g⟩

  ⟨*signature* `Prefs.abstract_bool_pref` 436h⟩

  ⟨*signature* `Prefs.abstract_string_pref` 436i⟩

  ⟨*signature* `Prefs.option_handlers` 437a⟩

  ⟨*signature* `Prefs.family` 437b⟩

  ⟨*signature* `Prefs.pref_error` 437c⟩

  ⟨*signature* `Prefs.resource_name` 437d⟩

  ⟨*signature* `Prefs.define` 437e⟩

F.9.9 gui/prefs.ml

⟨*function* Prefs.pref_error 438a⟩≡ (447)
```
  (* Generic report *)
  let pref_error msg =
    Frx_dialog.f Widget.default_toplevel (gensym "error")
       (s_ "Preference Error")
       msg
       (Predefined "") 0 [s_ "Ok"] |> ignore
```

⟨*function* Prefs.resource_name 438b⟩≡ (447)
```
  (* Converts an arbitrary string to a name suitable as a "global" resource *)
  let resource_name pref_name =
    let words = Mstring.split_str (function ' ' -> true | _ -> false) pref_name
    in
    (* for each words, remove non alpha-numerics *)
    (* in addition, make the each first characters capital *)
    let words' = List.map (fun word ->
      let buf = Bytes.create (String.length word) in
      let pos = ref 0 in
      for i = 0 to String.length word - 1 do
        if ('A' <= word.[i] && word.[i] <= 'Z') ||
           ('a' <= word.[i] && word.[i] <= 'z') ||
           ('0' <= word.[i] && word.[i] <= '9') then begin
          Bytes.set buf !pos word.[i];
          incr pos
      end;
      done;
      let x = Bytes.sub buf 0 !pos in
      begin
        try
      if 'a' <= Bytes.get x 0 && Bytes.get x 0 <= 'z' then
        Bytes.set x 0 (Char.chr (Char.code (Bytes.get x 0) + Char.code 'A' - Char.code 'a'));
        with
      Invalid_argument _ ->
        (* Strangely, x could be "". *) ()
      end;
      Bytes.to_string x ) words
    in
      "pref" ^ String.concat "" words'
```

⟨*constant* Prefs.class_name 438c⟩≡ (447)
```
  let _class_name = resource_name
    (* it is not correct but works *)
```

⟨*type* Prefs.pref_type *(*gui/prefs.ml*)* 438d⟩≡ (447)
```
  (*
   * Various predefined preference types
   *)
  type pref_type =
     Bool of bool ref
   | String of string ref
   | Int of int ref
   | Float of float ref
   | AbstractType of (Textvariable.textVariable -> unit) *
                     (Textvariable.textVariable -> unit)
                      (* init, set  as defined below *)
```

⟨*type* Prefs.pref (gui/prefs.ml) 439a⟩≡ (447)

```
(*
 * Support for interactive setting of a preference
 *)
type pref = {
  packed_widget : Widget.widget;        (* visual feedback *)
  pref_variable : Textvariable.textVariable;   (* placeholder for string
                         version of pref value, and
                         possibly "electric" change *)
  pref_type : pref_type;         (* internal definition *)
  pref_name : string;  (* internal name (shall not contain :) *)
  resource_name : string (* resource name (shall not contain :) *)
}
```

⟨*function* Prefs.init_pref 439b⟩≡ (447)

```
(*
 * Init the Tk variables in the pref editor from the internal
 * value of the preference (usually a reference)
 *)
let init_pref {pref_type = typ; pref_variable = v; _} = match typ with
   Bool r -> Textvariable.set v (if !r then "1" else "0")
 | String r -> Textvariable.set v !r
 | Int r ->  Textvariable.set v (string_of_int !r)
 | Float r ->  Textvariable.set v (string_of_float !r)
 | AbstractType(i,_) -> i v
```

⟨*function* Prefs.set_pref 439c⟩≡ (447)

```
(*
 * Set the internal preference value from the editor value (ie textvariable)
 * NOTE: basic predefined types do not allow extra code to run when the
 * value is modified.
 *)
let set_pref {pref_type = typ; pref_variable = v; _} = match typ with
   Bool r -> r := Textvariable.get v = "1"
 | String r -> r := Textvariable.get v
 | Int r ->
     let s = Textvariable.get v in
      begin try
        r := int_of_string s
      with Failure "int_of_string" ->
        pref_error (s_ "Not an integer: %s" s)
      end
 | Float r ->
     let s = Textvariable.get v in
     begin try
        r := float_of_string s
     with Failure "float_of_string" ->
        pref_error (s_ "Not a float: %s" s)
     end
 | AbstractType(_,s) -> s v
```

⟨*function* Prefs.load_pref 439d⟩≡ (447)

```
(*
 * Given the current resource database, set the internal and editor values
 * of the preference.
 *)
let load_pref pref =
  try
    let prefdata = Resource.get Widget.default_toplevel
      pref.resource_name pref.resource_name (* it is not correct but works *)
    in
    (* ONLY if non-empty ! *)
    if prefdata <> "" then begin
      Textvariable.set pref.pref_variable prefdata;
      set_pref pref
    end
  with
    Not_found -> () (* Never happen if database is complete *)
```

⟨*function* Prefs.save_pref 440a⟩≡ (447)

```
  (*
   * Adds the current pref value (from pref editor) to a preference table
   *)
  let save_pref add pref =
    add pref.resource_name (Textvariable.get pref.pref_variable)
```

⟨*function* Prefs.bool_pref 440b⟩≡ (447)

```
  (*
   * Building the preference manager for predefined preference types
   *)


  let bool_pref name r top =
    let v = Textvariable.create_temporary top in
    (* The frame is just to avoid expanding *)
    let f = Frame.create top [] in
    let w = Checkbutton.create f [Text name; Variable v] in
     pack [w][Side Side_Left; Anchor W; Fill Fill_X];
    let p =
      { pref_type = Bool r;
        pref_variable = v;
        packed_widget = f;
        pref_name = name;
        resource_name = resource_name name } in
    (* Automatically perform the preference change when you trigger the button *)
    Checkbutton.configure w [Command (fun () -> set_pref p)];
    p
```

⟨*function* Prefs.int_pref 440c⟩≡ (447)

```
  let int_pref name r top =
    let v = Textvariable.create_temporary top in
    let f,_e = Frx_entry.new_labelm_entry top name v in
    let p =
      { pref_type = Int r;
        pref_variable = v;
        packed_widget = f;
        pref_name = name;
        resource_name = resource_name name } in
    (* Automatically perform the preference change when you edit the entry *)
    (* NOTE: we have to use a "tracer" on the variable, since the user does *)
    (* not necessarily type Enter when editing is finished. OTOH, this will *)
    (* cause additionnal invocations during load_pref and init_pref *)
    let rec el () = Textvariable.handle v (fun () -> set_pref p; el()) in
    el(); p
```

⟨*function* Prefs.float_pref 440d⟩≡ (447)

```
  let float_pref name r top =
    let v = Textvariable.create_temporary top in
    let f,_e = Frx_entry.new_labelm_entry top name v in
    let p =
      { pref_type = Float r;
        pref_variable = v;
        packed_widget = f;
        pref_name = name;
        resource_name = resource_name name} in
    (* see above *)
    let rec el () = Textvariable.handle v (fun () -> set_pref p; el()) in
    el(); p
```

⟨*function* Prefs.string_pref 441a⟩≡ (447)

```
let string_pref name r top =
  let v = Textvariable.create_temporary top in
  let f,_e = Frx_entry.new_labelm_entry top name v in
  let p =
    { pref_type = String r;
      pref_variable = v;
      packed_widget = f;
      pref_name = name;
      resource_name = resource_name name } in
  (* see above *)
  let rec el () = Textvariable.handle v (fun () -> set_pref p; el()) in
  el(); p
```

⟨*function* Prefs.option_pref 441b⟩≡ (447)

```
let option_pref name (i, s, p) top =
  let v = Textvariable.create_temporary top in
  let f = Frame.create top [] in
  let l = Label.create f [Text name]
  and o,_ = Optionmenu.create f v p in
    pack [l;o][Side Side_Left];
  let p = {
    pref_type = AbstractType(i,s);
    pref_variable = v;
    packed_widget = f;
    pref_name = name;
    resource_name = resource_name name} in
  (* see above *)
  let rec el () = Textvariable.handle v (fun () -> set_pref p; el()) in
  el(); p
```

⟨*function* Prefs.abstract_bool_pref 441c⟩≡ (447)

```
(*
 * Like bool_pref, but with additional handling code
 *)

let abstract_bool_pref name i s top =
  let v = Textvariable.create_temporary top in
  (* The frame is just to avoid expanding *)
  let f = Frame.create top [] in
  let w = Checkbutton.create f [Text name; Variable v] in
   pack [w][Side Side_Left; Anchor W; Fill Fill_X];
  let p = {
    pref_type = AbstractType(i,s);
    pref_variable = v;
    packed_widget = f;
    pref_name = name;
    resource_name = resource_name name} in
  (* Automatically perform the preference change when you trigger the button *)
  Checkbutton.configure w [Command (fun () -> set_pref p)];
  p
```

⟨*function* Prefs.abstract_string_pref 442a⟩≡ (447)

```
  (*
   * Like string_pref, but with additional handling code
   *)
  let abstract_string_pref name i s top =
    let v = Textvariable.create_temporary top in
    let f,_e = Frx_entry.new_labelm_entry top name v in
    let p ={
      pref_type = AbstractType(i,s);
      pref_variable = v;
      packed_widget = f;
      pref_name = name;
      resource_name = resource_name name} in
    (* see above *)
    let rec el () = Textvariable.handle v (fun () -> set_pref p; el()) in
    el(); p
```

⟨*function* Prefs.option_handlers 442b⟩≡ (447)

```
  (*
   * Utility for option_pref
   *)

  let option_handlers mapping read_internal write_internal =
    let rev_mapping = List.map (fun (x,v) -> (v,x)) mapping in
    let init v =
      let current = read_internal() in
      let s =
        try
      List.assoc current mapping
        with
      Not_found ->
        match mapping with
          [] -> "undefined"
        | (_x,v)::_l -> v
      in
      Textvariable.set v s
    and set v =
      let current = Textvariable.get v in
      let value =
        try
      List.assoc current rev_mapping
        with
      Not_found ->
        match mapping with
          [] -> assert false
        | (x,_v)::_l -> x
      in
      write_internal value
    in
    init, set, List.map snd mapping
```

⟨*function* Prefs.load_file 442c⟩≡ (447)

```
  let load_file (f : Fpath.t) =
    (* It just loads the file as resource *)
    try
      Tkresource.readfile !!f Interactive
    with
      Protocol.TkError _ ->
        failwith (s_ "Can't open preference file: %s" !!f)
```

⟨*function* Prefs.save_file 443a⟩≡ (447)

```
let save_file prefmaps f =
  let delimiter = "!!! Don't edit below this line !!!" in
  try
    (* create $HOME/.mmm (by default) silently *)
    let prefdir = Filename.dirname f in
    if not (Sys.file_exists prefdir) then Munix.digdir prefdir 0o755;
    let oc = open_out (f ^ ".tmp") in
    try
      let ic = open_in f in
      try
    while true do
      let l = input_line ic in
      if l = delimiter then raise End_of_file
      else output_string oc (l ^ "\n")
    done
      with
    End_of_file ->
      close_in ic;
      raise End_of_file
    with
      Sys_error _
    | End_of_file ->
   (* the delimiter is found, no delimiter in the file
      or no pref file is found *)
    output_string oc (delimiter ^ "\n");
    List.iter (
    PrefMap.iter (fun name data ->
      output_char oc '*'; output_string oc name; output_char oc ':';
      output_string oc data; output_char oc '\n'))
      prefmaps;
    close_out oc;
    Unix.rename (f ^ ".tmp") f
  with Sys_error s ->
    pref_error (s_ "Can't open preference file: %s (%s)" f s)
```

⟨*type* Prefs.pref_family *(*gui/prefs.ml*)* 443b⟩≡ (447)

```
(* Builds a family of preferences *)
type pref_family =
  {family_widget: Widget.widget;    (* the main widget for this family *)
   family_init : unit -> unit;     (* init the display from memory *)
   family_save : unit -> string PrefMap.t; (* return current bindings *)
   family_load : unit -> unit;  (* loads from persistent storage *)
   family_title : string;
  }
```

⟨*function* Prefs.family 443c⟩≡ (447)

```
(* Computing a family from the predefined preference types *)
let family top title preff =
  let f =
    Frame.create_named top (Mstring.gensym "family")
      [Relief Sunken; BorderWidth (Pixels 1)] in
  (* create the widgets *)
  let prefs = List.map (fun p -> p f) preff in
  (* define the functions for the family *)
  let init _ = List.iter init_pref prefs
  and load () = List.iter load_pref prefs
  and save () =
    List.fold_right
      (fun pref map -> save_pref (fun k v -> PrefMap.add k v map) pref)
      prefs
      PrefMap.empty
  in
  (* initialize the text variables *)
  init();
  (* wrapping stuff *)
  let t = Label.create f [Text title] in
  pack [t][];
  pack (List.map (fun p -> p.packed_widget) prefs)
       [Fill Fill_X; Expand true; Anchor W];
  {family_widget = f; family_init = init;
   family_load = load; family_save = save;
   family_title = title}
```

⟨*function* Prefs.init 444⟩≡ (447)

```
(* This is the startup *)

let rec init (filename : Fpath.t ref) status interactive
    (mute : (unit -> unit) list) =
  let top = Toplevel.create_named Widget.default_toplevel "prefs"
                 [Class "MMMPrefs"] in
   Wm.title_set top (s_ "MMM Preferences");
   Wm.withdraw top;
   status := Some top;
   bind top [[], Destroy]
     (BindSet ([Ev_Widget],
           (fun ei -> if ei.ev_Widget = top then status := None)));

  let preffilev = Textvariable.create_temporary top in

  (* The menu bar *)
  let mbar = Frame.create_named top "menubar" [] in
  let file =
    Menubutton.create_named mbar "file" [Text (s_ "File"); UnderlinedChar 0] in
  pack [file][Side Side_Left];
  pack [mbar][Side Side_Top; Anchor W; Fill Fill_X];
  (* The window *)
  let hgroup = Frame.create_named top "panels" [] in
  (* section choice *)
  let sectionf = Frame.create_named hgroup "sections" [] in
  let buttonsf = Frame.create_named top "buttons" [] in
  pack [sectionf] [Side Side_Left; Fill Fill_Y];
  pack [hgroup] [Side Side_Top; Fill Fill_Both; Expand true];
  pack [buttonsf] [Side Side_Bottom];

  Textvariable.set preffilev (!! !filename); (* for the file selector *)

  (* We must load the file because some elements of the panel depend
     on resources defined in this file *)
  begin
    try load_file !filename
    with Failure s -> pref_error s
  end;

  (* Then we must do the mute stuff *)
  mute |> List.iter (fun f -> f());

  (* Then we can build the families *)
  let families = List.map (fun f -> f hgroup) interactive in
```

```
(* Then we do the interactive stuff *)
List.iter (fun f -> f.family_load ()) families;

let reset () =
  destroy top;
  status := None;
  init filename status interactive mute
in

(* select a preference file to load *)
let rec load () =
  Fileselect.f (s_ "Load a preference file")
    (function [] -> ()
            | [s] ->
       (* we must restart the panel, because resources
          may affect the displayed menus *)
        if Sys.file_exists s then begin
          destroy top;
          filename := Fpath.v s;
          init filename status interactive mute
        end
        else
          pref_error (s_ "%s : no such preference file" s)
            | _l -> raise (Failure "multiple selection"))
    (Filename.concat (Filename.dirname (Textvariable.get preffilev))
     "*")
    (Filename.basename (Textvariable.get preffilev))
    false
    false

(* select a new preference file to save in *)
and save_as () =
  Fileselect.f (s_ "Save preferences to file")
    (function
    [] -> ()
      | [s] ->
      Textvariable.set preffilev s;
      filename := Fpath.v s;
     begin
           try
      save_file (List.map (fun f -> f.family_save()) families) s;
      dismiss()
       with Failure s -> pref_error s
          end
      | _l -> raise (Failure "multiple selection"))
    (Filename.concat (Filename.dirname (Textvariable.get preffilev))
     "*")
    (Filename.basename (Textvariable.get preffilev))
    false
    false

(* save in the last defined preference file *)
and save () =
  try
    save_file (List.map (fun f -> f.family_save()) families)
          (Textvariable.get preffilev);
    dismiss()
  with
    Failure s -> pref_error s
and dismiss() =
  Wm.withdraw top

in

(* Fill in the menu *)
let mfile = Menu.create_named file "filemenu" [] in
  Menu.add_command mfile
     [Label (s_ "Load"); Command load; UnderlinedChar 0];
  Menu.add_command mfile
     [Label (s_ "Save"); Command save; UnderlinedChar 0];
  Menu.add_command mfile
     [Label (s_ "Save As"); Command save_as; UnderlinedChar 0];
  Menu.add_command mfile
      [Label (s_ "Dismiss"); Command dismiss; UnderlinedChar 0];
  Menubutton.configure file [Menu mfile];

(* Define the buttons *)
  let saveb = Button.create_named buttonsf "save"
    [Text (s_ "Save"); Command save]
  and resetb = Button.create_named buttonsf "reset"
    [Text (s_ "Reset"); Command reset]
  and dismissb = Button.create_named buttonsf "dismiss"
    [Text (s_ "Dismiss"); Command dismiss]
  in
  pack [saveb;resetb;dismissb][Side Side_Left; PadX (Pixels 20)];


let current = ref (List.hd families) in

let set_current f =
  Pack.forget [!current.family_widget];
  f.family_init();
  pack [f.family_widget]
       [Side Side_Top; Fill Fill_Both; Expand true];
  current := f in

let sectionv = Textvariable.create_temporary sectionf in
let selectors =
  List.map (fun f ->
    Radiobutton.create sectionf [
      Variable sectionv; Text f.family_title; Value f.family_title;
      Command (fun () -> set_current f)]
    )
  families;
in
pack selectors [Anchor W];

Textvariable.set sectionv !current.family_title;

set_current (List.hd families)
```

⟨*function* Prefs.define 446⟩≡ (447)

```
  (* Define a preference panel *)
  let define (filename : Fpath.t) interactive mute =
    let inited = ref None
    and current_file = ref filename in
    (function () ->
      match !inited with
      | Some w -> Wm.deiconify w
      | None -> (* we have been destroyed ! *)
        init current_file inited interactive mute
     )
```

```
⟨gui/prefs.ml 447⟩≡
  (* Preferences *)

  open Fpath_.Operators

  open I18n
  open Tk
  open Mstring


  ⟨function Prefs.pref_error 438a⟩

  ⟨function Prefs.resource_name 438b⟩

  ⟨constant Prefs.class_name 438c⟩

  ⟨type Prefs.pref_type (gui/prefs.ml) 438d⟩

  ⟨type Prefs.pref (gui/prefs.ml) 439a⟩


  ⟨function Prefs.init_pref 439b⟩
  ⟨function Prefs.set_pref 439c⟩

  ⟨function Prefs.load_pref 439d⟩
  ⟨function Prefs.save_pref 440a⟩


  ⟨function Prefs.bool_pref 440b⟩
  ⟨function Prefs.int_pref 440c⟩
  ⟨function Prefs.float_pref 440d⟩
  ⟨function Prefs.string_pref 441a⟩
  ⟨function Prefs.option_pref 441b⟩


  ⟨function Prefs.abstract_bool_pref 441c⟩
  ⟨function Prefs.abstract_string_pref 442a⟩

  ⟨function Prefs.option_handlers 442b⟩


  (*
   * Loading and saving preferences from a resource file
   *)

  module PrefMap = Map.Make(struct type t = string let compare = compare end)

  ⟨function Prefs.load_file 442c⟩
  ⟨function Prefs.save_file 443a⟩

  ⟨type Prefs.pref_family (gui/prefs.ml) 443b⟩

  ⟨function Prefs.family 443c⟩

  ⟨function Prefs.init 444⟩
  ⟨function Prefs.define 446⟩
```

F.9.10 gui/debug.ml

```
⟨gui/debug.ml 448a⟩≡

  open Protocol
  ⟨function Debug.active_cb 247a⟩

  ⟨function Debug.init 246i⟩
```

F.9.11 gui/history.mli

```
⟨gui/history.mli 448b⟩≡

  ⟨type History.history_entry 201a⟩

  ⟨type History.t 200b⟩

  val create: Document.id -> t

  val add: t -> Document.id -> string option -> unit

  val back: t -> (Document.id * string option) option
  val forward: t -> (Document.id * string option) option

  val contents: t -> entry list

  val set_current: t -> entry -> unit
```

F.9.12 gui/history.ml

```
⟨function History.contents 448c⟩≡                                        (449c)
  let contents h =
    let l = ref [] in
    let rec walk e =
      l := e :: !l;
      match e.h_next with
         None -> !l
        | Some e -> walk e
    in walk h.h_start

⟨function History.obsolete 448d⟩≡                                        (449c)
  (* Since a did may occur several times in the history, the list of
     obsolete entries is not simply the overwritten entries *)

  let obsolete current next =
    let kept = ref Document.DocumentIDSet.empty
    and forgotten = ref Document.DocumentIDSet.empty in
    let rec back e =
      kept := Document.DocumentIDSet.add e.h_did !kept;
      match e.h_prev with
        None -> ()
      | Some e -> back e in
    let rec forw e =
      forgotten := Document.DocumentIDSet.add e.h_did !forgotten;
      match e.h_next with
```

```
        None -> ()
      | Some e -> forw e in
    back current;
    forw next;
    Document.DocumentIDSet.diff !forgotten !kept
```

⟨*function* History.add 449a⟩≡ (449c)

```
  (* Add hinfo to the current point *)
  let add h did frag =
    (* Hack for the initial document *)
    if h.h_first then begin
      let newe = {h_did = did;
          h_fragment = frag;
          h_prev = None;
          h_next = None} in
        h.h_start <- newe;
        h.h_current <- newe;
        h.h_first <- false
        end
    else
      match h.h_current.h_next with
        None -> (* last in the list *)
      (* the new entry *)
      let newe = {h_did = did;
              h_fragment = frag;
              h_prev = Some h.h_current;
              h_next = None} in
      (* fix the linked list *)
      h.h_current.h_next <- Some newe;
      (* set the new current *)
      h.h_current <- newe
     | Some e -> (* adding in the middle of the list *)
      let newe = {h_did = did;
              h_fragment = frag;
              h_prev = Some h.h_current;
              h_next = None} in
      let dropped = obsolete newe e in
      h.h_current.h_next <- Some newe;
      h.h_current <- newe;
      Document.DocumentIDSet.iter (Gcache.remove h.h_key) dropped
```

⟨*function* History.set_current 449b⟩≡ (449c)

```
  let set_current h e =
    h.h_current <- e
```

⟨gui/history.ml 449c⟩≡

```
  (* History *)

  ⟨type History.history_entry 201a⟩

  ⟨type History.t 200b⟩

  ⟨function History.contents 448c⟩

  (* Did made obsolete by history overwriting *)
  ⟨function History.obsolete 448d⟩

  ⟨function History.add 449a⟩

  ⟨constant History.create 201b⟩
  ⟨function History.back 201c⟩

  ⟨function History.forward 201d⟩

  ⟨function History.set_current 449b⟩
```

F.9.13 gui/nav.mli

```
⟨signature Nav.display_headers 450a⟩≡                                    (450f)
  val display_headers : Document.handle -> unit

⟨signature Nav.copy_link 450b⟩≡                                          (450f)
  val copy_link : t -> Hyper.link -> unit

⟨signature Nav.save_link 450c⟩≡                                          (450f)
  val save_link : < Cap.network; ..> ->
    t -> (Unix.file_descr * bool) option -> Hyper.link -> unit

⟨signature Nav.add_user_navigation 450d⟩≡                                (450f)
  val add_user_navigation : string -> Viewers.hyper_func -> unit

⟨signature Nav.update 450e⟩≡                                             (450f)
  val update : < Cap.network; ..> ->
    t -> Document.id -> bool -> unit

⟨gui/nav.mli 450f⟩≡
  ⟨type Nav.t 35b⟩

  ⟨signature Nav.request 36b⟩

  ⟨signature Nav.display_headers 450a⟩

  ⟨signature Nav.copy_link 450b⟩
  ⟨signature Nav.save_link 450c⟩
  ⟨signature Nav.follow_link 35e⟩

  ⟨signature Nav.add_user_navigation 450d⟩

  ⟨signature Nav.make_ctx 36c⟩

  ⟨signature Nav.absolutegoto 34b⟩
  ⟨signature Nav.historygoto 201e⟩
  ⟨signature Nav.update 450e⟩

  ⟨signature Nav.dont_check_cache 236b⟩
```

F.9.14 gui/nav.ml

```
⟨exception Nav.Duplicate 450g⟩≡                                           (453)
  exception Duplicate of Url.t

⟨function Nav.nothing_specific 450h⟩≡                                     (453)
  (*
   * Three instances of this general mechanism : view, save, head
   *)
  let nothing_specific _nav _did _wr = raise Not_found
```

⟨*function* Nav.specific_viewer 451a⟩≡ (453)
```
  (* check the widget cache *)
  let specific_viewer addhist = fun nav did (wr : Www.request) ->
    let di = Gcache.find nav.nav_id did in
    if addhist then nav.nav_add_hist did wr.www_fragment;
    (* make it our current displayed document, since it is available *)
    nav.nav_show_current di wr.www_fragment
```

⟨*function* Nav.process_save 451b⟩≡ (453)
```
  (* Specific handling of "save" requests *)
  let process_save dest = fun _nav wr (dh : Document.handle) ->
    match dh.document_status with
      200 -> Save.transfer wr dh dest
    | n ->
      if wr.www_error#choose
          (s_ "Request for %s\nreturned %d %s.\nDo you wish to save ?"
                (Url.string_of wr.www_url) n (Http_headers.status_msg dh.dh_headers))
      then Save.transfer wr dh dest
      else Document.dclose true dh
```

⟨*function* Nav.display_headers 451c⟩≡ (453)
```
  (* Simple implementation of HEAD *)

  let display_headers (dh : Document.handle) =
    let mytop = Toplevel.create Widget.default_toplevel [] in
    Wm.title_set mytop
        (sprintf "HEAD %s" (Url.string_of dh.document_id.document_url));
    let hs =
      dh.dh_headers |> List.map (fun h -> Label.create mytop [Text h; Anchor W])
    in
    pack (List.rev hs) [Fill Fill_X];
    let b = Button.create mytop
              [Command (fun _ -> destroy mytop); Text "Dismiss"] in
    pack [b] [Anchor Center]
```

⟨*constant* Nav.process_head 451d⟩≡ (453)
```
  let process_head = fun _nav _wr dh ->
    Document.dclose true dh;
    display_headers dh
```

⟨*function* Nav.make_head 451e⟩≡ (453)
```
  (* But for head, we need to change the hlink *)
  let make_head (hlink : Hyper.link) =
    { hlink with h_method = HEAD; }
```

⟨*function* Nav.copy_link 451f⟩≡ (453)
```
  (* Copying a link to the X Selection *)
  let copy_link (nav : t) (h : Hyper.link) =
    try
      Frx_selection.set (Hyper.string_of h)
    with Hyper.Invalid_link _msg ->
      nav.nav_error#f (s_ "Invalid link")
```

⟨*constant* Nav.user_navigation 451g⟩≡ (453)
```
  let user_navigation = ref []
```

⟨*function* Nav.add_user_navigation 451h⟩≡ (453)
```
  let add_user_navigation (s : string) (f : Viewers.hyper_func) =
    user_navigation := (s,f) :: !user_navigation
```

⟨*function* Nav.save_link 452a⟩≡ (453)
```
  (* Simple wrappers *)
  let save_link (caps : < Cap.network; ..>)
       (nav : t) (whereto : (Unix.file_descr * bool) option) (lk : Hyper.link) :
      unit =
    request caps nav (process_save whereto) (true, id_wr, nothing_specific) lk
```

⟨*function* Nav.update 452b⟩≡ (453)
```
  let update (caps: < Cap.network; ..>)
      (nav : t) (did : Document.id) (nocache : bool) : unit =

    (* This gets called if answer is 200 but also 304 *)
    let process_update nav (wr : Www.request) (dh : Document.handle) =
      match dh.document_status with
        304 ->
      Cache.patch dh.document_id dh.dh_headers;
      Document.dclose true dh;
      begin try
        let di = Gcache.find nav.nav_id did in
        di#di_update
      with
        Not_found -> () (* weird *)
      end;
      wr.www_error#ok
        (s_ "Document %s has not changed.\n" (Url.string_of wr.www_url))
      | 200 | _ ->
          (* kill the previous displayed window *)
       Gcache.displace nav.nav_id did;
      (* we may have been redirected : check new did *)
      let oldurl = Url.string_of did.document_url in
      let newurl = Url.string_of dh.document_id.document_url in
      let add_hist = oldurl <> newurl in
      if add_hist then
        wr.www_error#ok (s_ "Document %s is relocated to:\n%s" oldurl newurl);
       wr.www_logging <- nav.nav_log;
       process_viewer add_hist (make_ctx caps) nav wr dh
    in

    try
      let doc = Cache.find did in
      try
        (* find the date of previous download, (or last-modified ?) *)
        let date_received = Http_headers.get_header "date" doc.document_headers in
        let follow_link =
          request caps nav process_update
          (false, (* we don't want to use cache here *)
           (* setup additional headers *)
           (fun wr ->
             wr.www_headers <-
               ("If-Modified-Since: "^date_received) :: wr.www_headers;
             if nocache
             then wr.www_headers <- "Pragma: no-cache" :: wr.www_headers;
             wr),
          nothing_specific)
        in
        follow_link (Hyper.default_link (Url.string_of did.document_url))
      with Not_found ->
        nav.nav_error#f ("Document has no Date: header.")
    with Not_found ->
     nav.nav_error#f (s_ "Document %s\nhas been flushed from cache"
                          (Url.string_of did.document_url))
```

⟨gui/nav.ml 453⟩≡

```
  open I18n
  open Tk

  (***************************************************************************)
  (* Prelude *)
  (***************************************************************************)
  (* Navigation *)

  (***************************************************************************)
  (* Types *)
  (***************************************************************************)

  ⟨type Nav.t 35b⟩

  ⟨exception Nav.Duplicate 450g⟩

  (***************************************************************************)
  (* Cache helpers *)
  (***************************************************************************)

  (* Important note: we assume two requests on the same url are identical
     (when we control emission of requests). This is not the case for
     POST requests, because we would need to check the POST data.
     This means that you can't post twice *simultaneously* on the same
     url. Proper fix: change the equality semantics of active cnx
   *)

  ⟨function Nav.dont_check_cache 236c⟩

  (***************************************************************************)
  (* request() *)
  (***************************************************************************)

  ⟨function Nav.request 36d⟩

  (***************************************************************************)
  (* XXX *)
  (***************************************************************************)

  ⟨function Nav.nothing_specific 450h⟩

  (***************************************************************************)
  (* Running viewers *)
  (***************************************************************************)

  ⟨function Nav.process_viewer 37c⟩

  ⟨function Nav.specific_viewer 451a⟩


  ⟨function Nav.process_save 451b⟩

  (***************************************************************************)
  (* Head ?? *)
  (***************************************************************************)

  ⟨function Nav.display_headers 451c⟩
  ⟨constant Nav.process_head 451d⟩

  ⟨function Nav.make_head 451e⟩

  (*****************************************************************************)
  (* Other handlers, less general *)
  (*****************************************************************************)

  ⟨function Nav.copy_link 451f⟩

  ⟨constant Nav.user_navigation 451g⟩
  ⟨function Nav.add_user_navigation 451h⟩

  ⟨function Nav.id_wr 40b⟩

  (*****************************************************************************)
  (* stdctx *)
  (*****************************************************************************)

  ⟨class Nav.stdctx 39⟩

  ⟨function Nav.make_ctx 38h⟩

  (*****************************************************************************)
  (* Follow/save links *)
  (*****************************************************************************)

  ⟨function Nav.save_link 452a⟩
  ⟨function Nav.follow_link 36a⟩

  (*
   * Other navigation functions
   *)

  ⟨function Nav.absolutegoto 35d⟩

  ⟨function Nav.historygoto 201f⟩

  ⟨function Nav.update 452b⟩
```

F.9.15 gui/mmmprefs.mli

```
⟨signature Mmmprefs.plug_applets 454a⟩≡                                    (454c)
  (* We need a more generic mechanism *)
  val plug_applets : (Widget.widget -> pref_family) -> unit

⟨signature Mmmprefs.f 454b⟩≡                                               (454c)
  val f : Fpath.t -> unit -> unit

⟨gui/mmmprefs.mli 454c⟩≡
  open Prefs

  ⟨signature Mmmprefs.plug_applets 454a⟩

  ⟨signature Mmmprefs.home 43g⟩

  ⟨signature Mmmprefs.f 454b⟩
```

F.9.16 gui/mmmprefs.ml

⟨*function* Mmmprefs.font_pref 455a⟩≡ (458g)
```
  (*
   * Font preference
   *)
  let font_pref title name top =
    let f = Frame.create top [] in
    let l = Label.create_named f "fontname" [Text title] in
    let f', v, i, s =
      Fontprefs.font_select f
        (fun () -> Styles.get_font name)  (* get from internal value *)
        (Styles.set_font name)  (* set internal value *)
    in
     pack [l;f'][Side Side_Left];
    (* map exceptions to error *)
    let i v = try i v with Failure s -> pref_error s
    and s v = try s v with Failure s -> pref_error s
    in
    let p = {
      pref_type = AbstractType(i,s);
      pref_variable = v;
      packed_widget = f;
      pref_name = title;
      resource_name = resource_name title} in
    p
```

⟨*constant* Mmmprefs.image_loading 455b⟩≡ (458g)
```
  (*
   * Image loading mode
   *)
  let image_loading =
    option_handlers
      [ Imgload.AfterDocManual, "After document, manual";
        Imgload.AfterDocAuto, "After document, automatic";
        Imgload.DuringDoc, "During document loading"]
      (fun () -> !Imgload.mode)
      (fun v -> Imgload.mode := v)
```

⟨*function* Mmmprefs.network 455c⟩≡ (458g)
```
  let network top =
    family top (s_ "Protocols") [
      string_pref "Proxy host" Http.proxy;
      int_pref "Proxy port" Http.proxy_port;
      bool_pref "Always Use Proxy" Http.always_proxy;
      bool_pref "HTTP Send Referer" Http.send_referer;
      string_pref "User Agent" Http.user_agent;
      int_pref "Timeout on headers (seconds)" Http.timeout;
      int_pref "Password lifetime (minutes)" Auth.lifetime;
      string_pref "Password save file" Auth.auth_file;
      abstract_string_pref "Local binaries path"
        Tk_file.pref_init Tk_file.pref_set
      ]
```

⟨*function* Mmmprefs.internal 455d⟩≡ (458g)
```
  let internal top =
    family top (s_ "Internal settings and debugging") [
      bool_pref "Strict encoding of Form field names" Urlenc.strict_form_standard;
      bool_pref "HTTP Requests" Http.verbose;
      int_pref "Internal buffer" Textw_fo.internal_buffer;
  (* always on for now    bool_pref "General trace" Log.debug_mode; *)
      bool_pref "Scheduler" Scheduler.debug;
      bool_pref "Cache debug" Cache.debug;
      bool_pref "Widget Cache debug" Gcache.debug;
      bool_pref  "HTML Display log" Html_disp.verbose;
      bool_pref "Table debug" Table.debug;
      bool_pref "Text fit debug" Fit.debug;
      bool_pref "Image loading debug" Img.ImageData.verbose;
      bool_pref "CamlTk Debug" Protocol.debug;
      ]
```

⟨*function* Mmmprefs.html 456a⟩≡ (458g)

```
  let html top =
    family top (s_ "HTML parsing and display") [
      option_pref "DTD" (dtd_i, dtd_s, dtd_p);
      bool_pref "Strict HTML lexing" Lexhtml.strict;
      bool_pref "Attempt tables" Html_disp.attempt_tables;
      bool_pref "Ignore relative TD width" Table.strict_32;
      bool_pref "Attempt smooth scroll" Htmlw.pscrolling;
      bool_pref "Frames as links" Htmlw.frames_as_links;
      abstract_string_pref "Background color"
        (fun v -> Textvariable.set v !Textw_fo.html_bg)
        (fun v ->
       let color = Textvariable.get v in
          Textw_fo.html_bg := color;
        (* transparent GIF hack, for the initial images *)
          Textvariable.set (Textvariable.coerce "TRANSPARENT_GIF_COLOR")
                           color;
              (* set the resource for each possible class of embedded windows *)
          Resource.add "*Html*Text.background" color WidgetDefault;
              Resource.add "*Html*Message.background" color WidgetDefault;
              Resource.add "*Html*Label.background" color WidgetDefault;
              Resource.add "*Html*Listbox.background" color WidgetDefault;
              Resource.add "*Html*Button.background" color WidgetDefault;
              Resource.add "*Html*Entry.background" color WidgetDefault;
              Resource.add "*Html*Menubutton.background" color WidgetDefault;
              Resource.add "*Plain*Text.background" color WidgetDefault
          );
      string_pref "Entry and Textarea color" Form.form_bg;
      bool_pref "Follow document colors" Textw_fo.usecolors;
      font_pref "Default font" "default";
      font_pref "<H1> font" "header1";
      font_pref "<H2> font" "header2";
      font_pref "<H3> font" "header3";
      font_pref "<H4> font" "header4";
      font_pref "<H5> font" "header5";
      font_pref "<H6> font" "header6";
      font_pref "Bold"   "bold";
      font_pref "Italic" "italic";
      font_pref "Fixed" "verbatim"
      ]
```

⟨*function* Mmmprefs.i18n 456b⟩≡ (458g)

```
  let i18n top =
    family top (s_ "Internationalization (Japanese)") [
      bool_pref "Ignore META charset" Htmlw.ignore_meta_charset
    ]
```

⟨*function* Mmmprefs.images 456c⟩≡ (458g)

```
  let images top =
    family top (s_ "Images") [
      bool_pref "No images at all" Imgload.no_images;
      option_pref "Image loading" image_loading;
         (* image_loading_i image_loading_s image_loading_p; *)
      int_pref "Max image connections" Img.ImageScheduler.maxactive;
      int_pref "Max image connections (same host)" Img.ImageScheduler.maxsamehost;
      float_pref "Gamma correction" Img.ImageData.gamma;
      string_pref "JPEG converter"  Img.ImageData.jpeg_converter
      ]
```

⟨*function* Mmmprefs.cache 457a⟩≡ (458g)

```
  let cache top =
    family top (s_ "Cache settings") [
      int_pref "Max number of documents"  Cache.max_documents;
      int_pref "Delete how much when full" Cache.cleann;
      bool_pref "Keep only history" Cache.history_mode;
      int_pref "Max cached widgets per window" Gcache.max_keep
      ]
```

⟨*function* Mmmprefs.progs 457b⟩≡ (458g)

```
  let progs top =
    family top (s_ "External programs") [
      string_pref "Mailto program" Mailto.mailer;
      string_pref "Hotlist program" Hotlist.program;
      string_pref "Printing program" Save.print_command;
      ]
```

⟨*function* Mmmprefs.misc 457c⟩≡ (458g)

```
  let misc top =
    family top (s_ "Misc. settings") [
      bool_pref "Use balloon helps" Balloon.flag;
      bool_pref "Use GIF animation" Img.gif_anim_load;
      bool_pref "Automatic GIF animation display" Imgload.gif_anim_auto
      ]
```

⟨*constant* Mmmprefs.appsys_plug 457d⟩≡ (458g)

```
  (* The default appsys preference only keeps track of
     the preference values, but does not allow changes
   *)
  let appsys_plug = ref (fun top ->
    let f = Frame.create top [Relief Sunken; BorderWidth (Pixels 1)] in
    let t = Label.create f [Text (s_ "Applets")] in
    let msg =
     Message.create f [Text (s_ "Applets are not available \
                                     in the native version")] in
    pack [t][Side Side_Top];
    pack [msg][Side Side_Bottom];
    (* we must keep track of applet preferences in the
       bytecode version : "Active" and "Paranoid" *)
    let active = ref false  and active_name = resource_name "Active"
    and paranoid = ref true and paranoid_name = resource_name "Paranoid" in
    let init () = ()  (* nothing special to be done *)
    and save () =
      List.fold_right
        (fun (name,value) map -> PrefMap.add name value map)
        [active_name, (if !active then "1" else "0");
         paranoid_name, (if !paranoid then "1" else "0")]
        PrefMap.empty
    and load () =
      List.iter (fun (name, setf) ->
        try
      let prefdata = Resource.get Widget.default_toplevel name name in
      setf prefdata
        with
      Not_found -> ())
        [active_name, (function data -> active := data = "1");
         paranoid_name, (function data -> paranoid := data = "1")]
    in
    {family_widget = f; family_init = init;
     family_save = save; family_load = load;
     family_title = s_ "Applets"})
```

⟨*function* Mmmprefs.plug_applets 458a⟩≡ (458g)

```
  let plug_applets f =
    appsys_plug := f
```

⟨*function* Mmmprefs.applets 458b⟩≡ (458g)

```
  let applets w = !appsys_plug w
```

⟨*function* Mmmprefs.reset_home 458c⟩≡ (458g)

```
  let reset_home () =
    home :=  Tkresource.string "wwwHome"
         (try Sys.getenv "WWW_HOME"
         with Not_found -> (Version.initurl (Lang.lang ())))
```

⟨*constant* Mmmprefs.mute 458d⟩≡ (458g)

```
  (* Internal preferences *)
  let mute = [
    reset_home;
    Fonts.reset;
    Viewers.reset;   (* viewers definition *)
    Glevents.reset;   (* bindings *)
    ]
```

⟨*constant* Mmmprefs.families 458e⟩≡ (458g)

```
  (* Interactive preferences *)
  let families = [ network; html; i18n; images; progs; cache; applets;
            misc; internal ]
```

⟨*function* Mmmprefs.f 458f⟩≡ (458g)

```
  (* main -> Mmm.initial_navigator -> <> *)
  let f (preffile : Fpath.t) =
    Prefs.define preffile families mute
```

⟨gui/mmmprefs.ml 458g⟩≡

```
  open Fpath_.Operators
  open I18n

  open Tk
  open Prefs

  (* MMM Preferences *)
```

⟨*function* Mmmprefs.font_pref 455a⟩

⟨*constant* Mmmprefs.image_loading 455b⟩

```
  (*
   * Choose from available DTDs for HTML parsing
   *)
  let dtd_i v =
    Textvariable.set v (Dtd.name !Dtd.current)
  and dtd_s v =
    Dtd.current :=
      try
        Dtd.get (Textvariable.get v)
      with
        Not_found -> Dtd.dtd32
  and dtd_p = Dtd.names()


  ⟨function Mmmprefs.network 455c⟩

  ⟨function Mmmprefs.internal 455d⟩

  ⟨function Mmmprefs.html 456a⟩

  ⟨function Mmmprefs.i18n 456b⟩

  ⟨function Mmmprefs.images 456c⟩


  ⟨function Mmmprefs.cache 457a⟩

  ⟨function Mmmprefs.progs 457b⟩

  ⟨function Mmmprefs.misc 457c⟩

  ⟨constant Mmmprefs.appsys_plug 457d⟩

  ⟨function Mmmprefs.plug_applets 458a⟩

  ⟨function Mmmprefs.applets 458b⟩


  ⟨constant Mmmprefs.home 43h⟩
  ⟨function Mmmprefs.reset_home 458c⟩


  ⟨constant Mmmprefs.mute 458d⟩

  ⟨constant Mmmprefs.families 458e⟩

  ⟨function Mmmprefs.f 458f⟩
```

F.9.17 gui/mmm.mli

```
⟨gui/mmm.mli 459⟩≡
  ⟨signature Mmm.user_file 208b⟩

  ⟨signature Mmm.initial_navigator 30a⟩
  ⟨signature Mmm.main_navigator 206e⟩

  ⟨signature Mmm.helpurl 209d⟩
  ⟨signature Mmm.initial_geom 208f⟩

  ⟨signature Mmm.add_user_menu 187g⟩
  ⟨signature Mmm.navigator 33b⟩
  ⟨signature Mmm.new_window_initial 207b⟩
  ⟨signature Mmm.new_window_sel 207c⟩
  ⟨signature Mmm.change_tachy 197b⟩
```

F.9.18 gui/mmm.ml

```
⟨gui/mmm.ml 460⟩≡
  open Common
  open Fpath_.Operators
  open I18n
  open Tk

  (*****************************************************************************)
  (* Prelude *)
  (*****************************************************************************)
  (* The navigation window *)

  (*****************************************************************************)
  (* Globals *)
  (*****************************************************************************)

  ⟨constant Mmm.hotlist 206a⟩
  ⟨constant Mmm.helpurl 209e⟩
  ⟨constant Mmm.initial_page 33c⟩
  ⟨constant Mmm.initial_geom 208g⟩

  (*****************************************************************************)
  (* Helpers *)
  (*****************************************************************************)

  ⟨constant Mmm.home 208d⟩

  ⟨function Mmm.user_file 208c⟩

  ⟨constant Mmm.preferences 207g⟩

  (*****************************************************************************)
  (* Tachy *)
  (*****************************************************************************)

  ⟨constant Mmm.container_frame 198⟩
  ⟨constant Mmm.tachy_maker 197d⟩

  ⟨function Mmm.change_tachy 197c⟩

  ⟨function Mmm.start_tachy 197e⟩

  (*****************************************************************************)
  (* Display/undisplay *)
  (*****************************************************************************)

  (* Switching current viewers in the browser *)
  ⟨function Mmm.undisplay 38f⟩
  ⟨function Mmm.display 38b⟩

  (*****************************************************************************)
  (* Helpers *)
  (*****************************************************************************)
```

```
⟨function Mmm.quit 45g⟩

⟨constant Mmm.user_menus 187h⟩
⟨function Mmm.add_user_menu 187i⟩

(*****************************************************************************)
(* navigator() *)
(*****************************************************************************)

⟨constant Mmm.navigators 206g⟩

⟨function Mmm.navigator 34a⟩


⟨function Mmm.new_window_initial 207d⟩

⟨function Mmm.new_window_set 207e⟩

(*****************************************************************************)
(* initial_navigator() *)
(*****************************************************************************)

⟨constant Mmm.main_navigator 206f⟩

⟨function Mmm.initial_navigator 33a⟩
```

F.9.19 gui/cci.ml

⟨*function* Cci.handler 461⟩≡ (463a)

```
  (* CCI was cool, but nobody implements it anymore. More over,
   * it's trivial to fork mmm_remote and let the protocol be managed
   * by it *)

  let handler (caps : < Cap.network ; .. >) (fd : Unix.file_descr) (line : string)
     =
    let len = String.length line in
    if len > 4 && String.sub line 0 4 = "GET " then begin
      let url = String.sub line 4 (len - 4) in
      match !Mmm.main_navigator with
      | None ->
          Munix.write_string fd "No main navigator\n";
          Unix.close fd
      | Some nav ->
          Nav.save_link caps nav
            (Some (fd, true))
            { h_uri = url; h_context = None; h_method = GET; h_params = [] }
    end
    else if len > 5 && String.sub line 0 5 = "GETB " then begin
      let url = String.sub line 5 (len - 5) in
      match !Mmm.main_navigator with
      | None ->
          Munix.write_string fd "No main navigator\n";
          Unix.close fd
      | Some nav ->
          Nav.save_link caps nav
            (Some (fd, false))
            { h_uri = url; h_context = None; h_method = GET; h_params = [] }
    end
    else if len > 5 && String.sub line 0 5 = "HEAD " then begin
      let url = String.sub line 5 (len - 5) in
      match !Mmm.main_navigator with
      | None ->
          Munix.write_string fd "No main navigator\n";
          Unix.close fd
      | Some nav ->
          Nav.save_link caps nav
            (Some (fd, true))
            { h_uri = url; h_context = None; h_method = HEAD; h_params = [] }
    end
    else if len > 8 && String.sub line 0 8 = "DISPLAY " then begin
      let url = String.sub line 8 (len - 8) in
      Unix.close fd;
      ignore (Mmm.navigator caps false (Lexurl.make url))
    end
    else begin
      (* assume DISPLAY (backward compatibility) *)
      Unix.close fd;
      ignore (Mmm.navigator caps false (Lexurl.make line))
    end
```

⟨*function* Cci.init 462⟩≡ (463a)

```
  (* External requests *)
  let init (caps : < Cap.network ; .. >) =
    let file = Mmm.user_file "remote" in
    try
      let socket = Unix.socket PF_UNIX SOCK_STREAM 0 in
      begin
        try Unix.bind socket (ADDR_UNIX !!file) with
        | _ ->
            if not (Sys.file_exists !!file) then raise Not_found;
            begin
              match
                Frx_dialog.f Widget.default_toplevel (Mstring.gensym "confirm")
                  (s_ "Confirm")
                  (s_
                     "%s already exists. This may mean that there is another MMM \
                      already running. Do you want to remove this file and \
                      create again ? (Note that you must be sure there is no \
                      other MMM with -external option)"
                     !!file)
                  (Predefined "question") 0
                  [ s_ "Yes"; s_ "No, I give up to use -external option" ]
              with
              | 0 ->
                  Unix.unlink !!file;
                  Unix.bind socket (ADDR_UNIX !!file)
              | _ -> raise Exit
            end
      end;

      Unix.listen socket 5;
      Fileevent.add_fileinput socket (fun () ->
          try
            let fd, _ = Unix.accept socket in
            handler caps fd (Low.read_line fd)
          with
          | _ -> ());
      at_exit (fun () -> Msys.rm !!file)
    with
    | e -> Error.f (s_ "Can’t initialize %s\n%s" !!file (Printexc.to_string e))
```

⟨gui/cci.ml 463a⟩≡

```
  open Fpath_.Operators
  open I18n

  (* pad: CCI = Common Client Interface? *)

  ⟨function Cci.handler 461⟩

  ⟨function Cci.init 462⟩
```

F.10 main/

F.10.1 main.ml

⟨main.ml 463b⟩≡

```
  open Common
  open Fpath_.Operators

  (*****************************************************************************)
  (* Prelude *)
  (*****************************************************************************)
  (* The MMM Web Browser *)

  (*****************************************************************************)
  (* Types and constants *)
  (*****************************************************************************)
  ⟨type Main.caps 29a⟩

  (*****************************************************************************)
  (* Helpers *)
  (*****************************************************************************)

  ⟨function Main.safe_loop 40d⟩

  ⟨function Main.localize 31e⟩

  (*****************************************************************************)
  (* The options *)
  (*****************************************************************************)

  (*****************************************************************************)
  (* Main entry point *)
  (*****************************************************************************)

  ⟨constant Main.usage_str 15a⟩

  ⟨function Main.main 29c⟩

  ⟨function Main.postmortem 244a⟩

  ⟨toplevel Main._1 243h⟩
```

F.10.2 main_remote.ml

```
⟨main_remote.ml 464a⟩≡
  (* Talk to an mmm master *)

  type caps = < Cap.network >

  ⟨function Main_remote.request 219f⟩

  ⟨function Main_remote.main 220a⟩

  ⟨toplevel Main_remote._1 220b⟩
```

F.11 extensions/

F.11.1 extensions/audio.ml

```
⟨extensions/audio.ml 464b⟩≡
  open Safe418mmm

  open Tk
  open Hyper
  open Viewers
  open Document

  module Provide = struct
    let capabilities = Capabilities.get()
    end
  module Mmm = Get(Provide)


  ⟨function Audio.fake_embed 219a⟩
  ⟨toplevel Audio._1 219b⟩
```

F.11.2 extensions/images.ml

```
⟨function Images.images 464c⟩≡                                        (465e)
  let images lexbuf =
    let uris = ref [] in
    try
      let lexer = ParseHTML.sgml_lexer Dtd.dtd32 in
      while true do
        try
          let _,_,tokens,loc = lexer lexbuf in
        List.iter (function
           OpenTag {tag_name = "img"; attributes = attrs} ->
             begin try
          uris := List.assoc "src" attrs :: !uris
             with Not_found -> ()
             end
         | EOF -> raise End_of_file
         | _ -> ())
            tokens
        with
          Html_Lexing _ -> ()
        | Invalid_Html _ -> ()
      done;
      !uris
    with
      End_of_file -> List.rev !uris
```

⟨*function* Images.show_images 465a⟩≡ (465e)

```
  (* Pops up a dialog box with the list of image URLs *)
  let show_images ctx l =
    let w = Applets.get_toplevel_widget []
    and base = Url.string_of (ctx#base.document_url)
    in
    Wm.withdraw w;
    Frx_req.open_list "Display Images" l
      (fun uri ->
        let link =
      {h_uri = uri; h_context = Some base; h_method = GET; h_params = []} in
          ctx#goto link)
      (fun _ -> destroy w)
```

⟨*function* Images.f 465b⟩≡ (465e)

```
  (* When the menu item is activated, this function is called :
     we're interested mostly in the URL of the currently displayed document,
     but the ctx will be used later so we can trigger new navigation functions
     on the URLs of the in-lined images.
     What we do is request a copy of this document, on which we run an HTML
     lexer.
   *)
  let f ctx =
    let cont = {
      document_process = (fun dh ->
        let lexbuf = Lexing.from_function
                     (fun buf n -> dh.document_feed.feed_read buf 0 n) in
        let l = images lexbuf in
          dclose true dh;
          show_images ctx l);
      document_finish = (fun _ -> ())
      } in
    let link = Hyper.default_link (Url.string_of ctx#base.document_url) in
    Net.retrieve link cont
```

⟨*toplevel* Images._1 465c⟩≡ (465e)

```
  let _ = Mmm.add_user_menu "In-lined images" f
```

⟨*toplevel* Images._2 465d⟩≡ (465e)

```
  let _ = Applets.register "main"
      (fun f ctx ->
        pack [Label.create f [Text "Menu User/images installed"]][])
```

⟨extensions/images.ml 465e⟩≡

```
  open Safe418mmm

  (* This module demonstrates
      - how to add an user menu
      - how to call the HTML lexer
  *)

  module Provide = struct
    let capabilities = Capabilities.get()
    end

  module Net = Retrieval(Provide)
  module Mmm = Get(Provide)

  open Tk
  open Net
  open Html
  open Document
  open Feed
  open Hyper
  open Viewers

  ⟨function Images.images 464c⟩

  ⟨function Images.show_images 465a⟩

  ⟨function Images.f 465b⟩

  ⟨toplevel Images._1 465c⟩


  ⟨toplevel Images._2 465d⟩
```

F.11.3 extensions/remove_simple_table.ml

```
⟨function Remove_simple_table.log 466a⟩≡                              (468b)
  let log s = try prerr_endline s with _ -> ()

⟨type Remove_simple_table.table_token 466b⟩≡                          (468b)
  type table_token =
      ChildTable of Html.token list
    | Token of Html.token

⟨type Remove_simple_table.rst_env 466c⟩≡                              (468b)
  type rst_env = {
      mutable tokens : table_token list;
      mutable trs : int;
      mutable tds : int
    }

⟨function Remove_simple_table.remove_simple_table 466d⟩≡              (468b)
  let remove_simple_table parentf =
    let stack = ref [] in

    let push_tbl tbl = stack := tbl :: !stack in
    let pop_tbl () =
     match !stack with
     | t :: tl -> stack := tl; t
     | _ -> assert false in
    let head_stack () = List.hd !stack in
    let empty_stack () = !stack = [] in

    let flush_childtable tkns =
      if empty_stack () then List.iter parentf tkns
      else begin
        let top = head_stack () in
        top.tokens <- top.tokens @ [ChildTable tkns]
      end
    in

    fun tkn -> match tkn with
      EOF ->
        while not (empty_stack ()) do
      log "EOFflush";
      let tbl = pop_tbl () in
      flush_childtable
        (List.fold_right (fun xtkn st ->
          match xtkn with
            Token tkn -> tkn :: st
          | ChildTable tkns -> tkns @ st) tbl.tokens [])
        done;
        parentf EOF
    | OpenTag {tag_name = "table"} ->
        log "ENTER";
        let tbl = {tokens= [Token tkn]; trs= 0; tds= 0} in
        push_tbl tbl;
    | CloseTag "table" when not (empty_stack ()) ->
        let tbl = pop_tbl () in
        log "REMOVE";
        let tokens =
      if tbl.trs <= 1 && tbl.tds <= 1 then begin
        log "ERASE";
        let tokens =
         (* remove table, tr, td *)
         List.fold_right (fun xtkn st ->
            match xtkn with
          Token tkn -> begin
            match tkn with
              OpenTag {tag_name= "table"}
            | OpenTag {tag_name= "tr"}
            | OpenTag {tag_name= "td"}
            | CloseTag "table"
            | CloseTag "tr"
            | CloseTag "td" -> st
            | _ -> tkn :: st
          end
            | ChildTable tkns ->
            tkns @ st) tbl.tokens []
        in
        [OpenTag {tag_name="br"; attributes=[]};
          CloseTag "br";
          PCData "[[" ] @ tokens @
        [ PCData "]]";
          OpenTag {tag_name="br"; attributes=[]};
          CloseTag "br" ]
      end else begin
        tbl.tokens <- tbl.tokens @ [Token tkn];
        List.fold_right (fun xtkn st ->
          match xtkn with
            Token tkn -> tkn :: st
          | ChildTable tkns -> tkns @ st) tbl.tokens []
      end
        in
        flush_childtable tokens
    | _ when not (empty_stack ()) ->
        let tbl = head_stack () in
        begin
      match tkn with
        OpenTag {tag_name = "td"} ->
          tbl.tds <- tbl.tds + 1
      | OpenTag {tag_name = "tr"} ->
```

```
          tbl.trs <- tbl.trs + 1
      | _ -> ()
        end;
        tbl.tokens <- tbl.tokens @ [Token tkn]
    | _ ->  (* !stack = [] *)
        parentf tkn
```

⟨*toplevel* `Remove_simple_table._1` 468a⟩≡ (468b)

```
  let _ =
    Mmm.add_html_filter remove_simple_table
```

⟨`extensions/remove_simple_table.ml` 468b⟩≡

```
  open Safe418mmm

  ⟨function Remove_simple_table.log 466a⟩

  (* an example of html filter *)

  (* This example removes the tables with atmost one <TD> (<TR> also) tag. *)
  (* This reduces the widgets creations... *)

  module Provide = struct
    let capabilities = Capabilities.get()
    end

  module Mmm = Get(Provide)

  open Html

  ⟨type Remove_simple_table.table_token 466b⟩

  ⟨type Remove_simple_table.rst_env 466c⟩

  ⟨function Remove_simple_table.remove_simple_table 466d⟩

  ⟨toplevel Remove_simple_table._1 468a⟩
```

F.11.4 extensions/tachy_aftermmm.ml

⟨*constant* `Tachy_aftermmm.tachy_data` 468c⟩≡ (470f)

```
  (* inside bitmap, circle is in +10+6 +47+43, radius 18.5 *)

  let tachy_data = "GIF\056\055a\058\000\060\000\165\000\000\168\168\168\
  \168\152\176\152\152\168\152\152\152\152\136\160\136\152\152\136\
  \136\152\128\136\128\128t\152\144\172\200\136\152\184\144\132\
  \168\128\132\152\136\136\136\000\000\000\248\252\248\216\216\216\
  \040\040\040p\136\136\184\184\184\200\200\200XXX\128\
  txht\152pt\136\040\044\040ptphd\
  \144\232\232\232\096d\136\224pp\240xx\096dp\
  \096\096h\096TpPdpHXxXTX\000\
  \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
  \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
  \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
  \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\
  \000\000\000\000\000\000\000\000\000\000\000\000\000\044\000\000\
  \000\000\058\000\060\000\000\006\254\064\128\016\016\032\026\139\
  \001\001R\201\020\052\007\004\168\148P\160Z\013W\002\
  \214\096\056p\187\136\195\016\144\024\011\145\000\005\064\176\
  n\179\005\010\167\160\048\159\091\011\139\042\190\096\224\243\
\191\006\012\129\006gCh\135JDJ\003I\002\003\
\142\142QVPzY\091\093\152\007aae\134G\
FjEjr\002\013\166\167\014\015\003\003\166\016\149\
\123\095\127\092\017\180\180\018\008\135G\137I\188N\172\
\167\192\013\169\171\196\019\020\174Z\092\201\094\179\155\018\
\016bf\159\163n\193\214\195\196\003c\016\125\221\128\
\018\017\130\183\220\185\136\189\214\193\019c\196\210\016\220\
\153\096\017\155\007\016\018\159h\161m\021\214\171\210\254\
\237\128\004\025\136pk\028\046\093Kx\057\216\215\224\
\215\191\135f\038L\128\006F\211\051w\238\236\009\201\
G\141\205\062\134\013\033\138\148\056\193A\061\003\018P\
\034\040\184\242\160\057\038\251\022\134\020\057\178d\003w\
\154r\174\180\032\129\231\254\180\053N\000\124\020V\129\
\166Q\000\013\000H\172\215\146\165\132\011\188\020\245\026\
j\234\232Q\137\020\142\021\228\185\019\003\040\055k\128\
i\179j\212XV\008MW\094\144\128\033a\147\096\
d\173f\061\203\179\046\006\011\094G\005u\034\054\174\
\220\179\005\215b\192\000\149I\163\095\167\252Z\053\155\
\017\239\096\188\096\229\000\051\154\161B\006\162E\043g\
\016\057\023\130\096\182l\123AB\156\148f\209\152B\
\133V\230\124\204\029\094\199\026\246J\062\053\022b\131\
\213\183\047\035\205\160\014\226D\140\131\217\014\062\012i\
\178\233\210\167\027TX\094T\036F\008\176\055x\037\
\053\199\184\200\125B\150\011e\222\220\057\004\014\194\007\
c\024\029\201\186b\185\192\053\096P\031\091\014\029\005\
\180\207\251\061\037\030C\007\012Q\034Aq\192\191\182\
\252\163\252\057\032\221\006\026\016\232\004\029\008\146T\026\
Y\020\040F\210\096\234\221\167\159\021\166\248\247\159Q\
\030d\184\030\129\004\254\190gG\001\236\092hU\134\
\030\220w\095\129\146T\146\141\136F\125\144\225\007\028\
\022\168\001\130w\172\200\162H\030\124\160\035\123\004\226\
\247J\020\033\222\248\208\139\048\202\216\225\029\147\244\035\
\228\063\057\234\184\161\145Hb\001\162\133K\142A\164\
\145\026h\000\130\030yP\177\128\146U\154\049\064\147\
E\174\215\001\138\063bAe\152\058\234\024c\150Y\
\190\146\007\023aJ\211\230\142\049\018\216\129\022\124\126\
\161E\131u\010q\167\140Zn\000\130\150\092\250\209\
\013\004CH\180\036\153\031\152\169\193\153g\130\096\201\
\050\006\048\042\145r\055\054\064\036\129Z\194y\232\150\
s\002\002\008\004\019\008\195\168\136\015\056\224b\145\049\
V\010\130\165\126r\161\147\038\020\152D\146\124\020p\
\224\171\171\160\142\170\229\172\179\202\194\135\064\040\037\139\
\042I\128\198\229\235\003\190B\160\231\164\212\018\171\037\
\001\007\044\019FE\059\045kV\179\052\245\234\235\184\
\193\134\042\236\172\254\033\152\250EJ\226\180\004\193\092\
\240be\198\092\227\214\203\193\180\211\018\171\239\023\183\
\058\211\020\094\238\192\219\171\059\037\061\128\145\189\028\064\
\064\232\185\250\130\016\130\008\233\014\034N\178N\053u\
\193\187\024cT\176\059\245\026\028\043\181\178\018\059\002\
\008\035W\196mO\091\161\252\218\093\024\060\183q\189\
\160\194i\238\176\163\058\012\177\195\223\012Rq\096\225\
\133\055\152\137O\230\009\050\008\033\207\058\178\209\034\204\
\227\239V\044\059\198\242\134\016\010\045\244\204\013\219\252\
\240\213\019\167\228\212Zj\129\134\129\207\246\133\045c\
\208\213\154\171\239\200h\147\012\194\201\092\245t\151\219\
\043\175\039\055\150o\202\204\176\190W\091\045\194\008\005\
\169\180\179\096\245\213\007\052\221\050RJ\052\205F\139\
\188\055\201I\091\244o\211\226E\135\129\212\132\046l\
\054\222\016g\254\240\008\123km\049h\128\127\029\054\
\132b\247\136e\165\151\039\158\054\218\123\039\141rW\
  \043\199\061\054j\225\049\139\154\058\186\154g\206\249\230\
  \059w\029\184\224\165\019\046k\209j\035\205\058\231\035\
  \144\240x\220\245\209\142e\161YZ\091\245\213\173o\
  \222\250\008\033\148\192\179\232\096\015\062\183\233\031\223\173\
  \058\227\199\239MB\242\176C\030\029\143\132\203L\181\
  \195\196RO\061\242\123g\159\253S\219\255N\250\137\
  \148\135\140\184\218\229\163\095\242\146W\002\018\004\001\000\
  \059\
  "
```

⟨*constant* Tachy_aftermmm.park_data 470a⟩≡ (470f)

```
  let park_data =
  "#define break_width 15
  #define break_height 11
  static char break_bits[] = {
     0x0c, 0x18, 0xf4, 0x17, 0x3a, 0x2e, 0xba, 0x2d, 0xb9, 0x4d, 0x3d, 0x5e,
     0xb9, 0x4f, 0xba, 0x2f, 0xba, 0x2f, 0xf4, 0x17, 0x08, 0x08};
  "
```

⟨*constant* Tachy_aftermmm.pi 470b⟩≡ (470f)

```
  let pi = 3.1415926
```

⟨*constant* Tachy_aftermmm.log10 470c⟩≡ (470f)

```
  let log10 = log 10.0
```

⟨*function* Tachy_aftermmm.create_tachy 470d⟩≡ (470f)

```
  let create_tachy top =
    let o = new default_tachy top in
    o#start;
```

⟨*toplevel* Tachy_aftermmm._1 470e⟩≡ (470f)

```
  let _ =
    let top = Applets.get_toplevel_widget [] in
    Wm.withdraw top;
    begin match Frx_dialog.f top (Mstring.gensym "foo")
          "Tachy test" "Use this aftermmm tachymeter"
      (Tk.Predefined "question") 1 ["Yes"; "No"] with
      0 -> Mmm.set_tachy create_tachy
    | _ -> ()
    end;
    destroy top
```

⟨extensions/tachy_aftermmm.ml 470f⟩≡

```
  open Safe418mmm

  open Tk

  module Provide = struct
    let capabilities = Capabilities.get()
    end

  module Mmm = Get(Provide)

  (* Tachymeter *)

  (* gif is 58x60 *)
```

  ⟨*constant* Tachy_aftermmm.tachy_data 468c⟩

  ⟨*constant* Tachy_aftermmm.park_data 470a⟩

⟨*constant* Tachy_aftermmm.pi 470b⟩
⟨*constant* Tachy_aftermmm.log10 470c⟩

```
class default_tachy (top : Widget.widget) =
 object (self)
  (* val top = top *)
  val mutable canvas = top (* dummy initialisation *)
  val mutable alive = false

  (* Various components of the canvas, all with dummy init values *)
  val mutable i_park = Tag "none"
  val mutable kilos = Tag "none"
  val mutable aig = Tag "none"
  val mutable pendings = Tag "none"


  (* this one is private *)
  method start =
    let c =
      Canvas.create_named top "tachymeter"
       [Width (Pixels 56); Height (Pixels 60);
         BorderWidth (Pixels 0);
         HighlightThickness (Pixels 0);
         TakeFocus true (* pl3 fix *)] in
    (* Use colors so that images are not transparent *)
    (*
    let tachy_image =
      begin
       try
      let bgc = Tk.cget c CBackground in
      Protocol.tkEval
        [|Protocol.TkToken "set";
          Protocol.TkToken "TRANSPARENT_GIF_COLOR";
          Protocol.TkToken bgc |]; ()
       with _ -> ()
      end;
      *)
    let tachy_image = Frx_misc.create_photo [Data tachy_data]
    and park_image =
      Imagebitmap.create [Data park_data; Foreground Red] in

    i_park <-
      Canvas.create_rectangle c
     (Pixels 50) (Pixels 4)
     (Pixels 53) (Pixels 7) [FillColor Black];

    kilos <-
      Canvas.create_text c (Pixels 28) (Pixels 52) [Text "0"; Font "-adobe-helvetica-medium-r-*-*-8-*-*-*-*-*-*

    aig <-
      Canvas.create_line c [Pixels 27; Pixels 25; Pixels 27; Pixels 43]
                        [Width (Pixels 2)];
    pendings <-
      Canvas.create_text c (Pixels 52) (Pixels 39) [Text "0"; Font "-adobe-helvetica-medium-r-*-*-8-*-*-*-*-*-*

    let i_tachy =
      Canvas.create_image c (Pixels 56) (Pixels 0)
       [ImagePhoto tachy_image; Anchor NE]
    in

    Canvas.lower_bot c pendings;

    (* All other items must be put above the background image *)
    List.iter (fun i -> Canvas.raise_above c i i_tachy)
      [kilos; aig; i_park];

    bind c [[], Destroy] (BindSet ([], (fun _ -> alive <- false)));

    (* These bindings are specific to the applet version *)
    bind c [[], ButtonPressDetail 1]
      (BindSet ([], (fun _ -> Mmm.new_window_initial (); ())));
    bind c [[], ButtonPressDetail 2]
      (BindSet ([], (fun _ -> Mmm.new_window_sel (); ())));

    alive <- true;
    pack [c][];
    canvas <- c

  val mutable last_speed = 0.
  val mutable last_total = 0
  val mutable idle = false

  method update speed total =
    if speed = 0.0 then begin
      if not idle then begin
    Canvas.configure_rectangle canvas i_park [FillColor Black;
                          Outline Black];
    idle <- true
      end
    end
    else begin
      Canvas.configure_rectangle canvas i_park [FillColor Green;
                         Outline Green];
      idle <- false
    end;
    if total <> last_total then
      Canvas.configure_text canvas kilos [Text (string_of_int total)];
    last_total <- total;
    let speed = if speed = 0. then 0. else log speed in
       (* Smooth *)
    let speeds = (last_speed +. speed) /. 2. in
    if abs_float (speeds -. last_speed) > 0.1 then begin
      last_speed <- speeds;
      let v = speeds /. log10 in
      let angle = v /. 4.0 *. pi in
      let angle = if angle < 0.1 then 0.0 else angle in
      let x = 27.0 -. (sin angle *. 18.5)
      and y = 25.0 +. (cos angle *. 18.5) in
      Canvas.coords_set canvas aig
    [Pixels 27; Pixels 25;
      Pixels (truncate x); Pixels (truncate y)];
      update_idletasks()
    end

  method report_cnx n =
    if Winfo.exists canvas then
      if n = 0 then begin
    Canvas.configure_text canvas pendings [Text ""];
```

```
        Canvas.lower_bot canvas pendings
       end
       else begin
     Canvas.configure_text canvas pendings
       [Text (string_of_int n)];
        Canvas.raise_top canvas pendings
       end

   method report_busy busy =
     if Winfo.exists canvas then
       if busy then begin
        Canvas.lower_bot canvas pendings;
     Canvas.configure_rectangle canvas i_park [FillColor Red;
                           Outline Red];
     update_idletasks()
       end
       else begin
        Canvas.raise_top canvas pendings;
     Canvas.configure_rectangle canvas i_park [FillColor Black;
                           Outline Black]
       end

   method report_traffic tick_duration bytes_read sample_read =
     if alive then
       self#update (float sample_read *. 1000. /. float tick_duration)
     bytes_read

   method quit =
     alive <- false;
     destroy canvas

 end
```

⟨*function* `Tachy_aftermmm.create_tachy` 470d⟩
```
   (o :> Mmm.tachymeter)
```

⟨*toplevel* `Tachy_aftermmm._1` 470e⟩

F.11.5 extensions/tachy_space.ml

⟨*constant* `Tachy_space.mpoly_data` 473⟩≡ (474i)
```
 (* Tachymeter *)

 let mpoly_data =
 [5.684359, -36.000000;
 14.760086, -36.000000;
 19.000000, -20.736308;
 23.239914, -36.000000;
 31.760086, -36.000000;
 36.000000, -20.736308;
 40.239914, -36.000000;
 46.315641, -36.000000;
 37.760086, -5.200000;
 30.239914, -5.200000;
 26.000000, -20.463692;
 21.760086, -5.200000;
 14.239914, -5.200000]
```

⟨*type* Tachy_space.vector 474a⟩≡ (474i)
```
  type vector = float * float * float
```

⟨*type* Tachy_space.matrix 474b⟩≡ (474i)
```
  type matrix = vector * vector * vector
```

⟨*function* Tachy_space.matrix_vector 474c⟩≡ (474i)
```
  let matrix_vector ((a11,a21,a31), (a12,a22,a32), (a13,a23,a33)) (x,y,z) =
    (a11*.x+.a12*.y+.a13*.z, a21*.x+.a22*.y+.a23*.z, a31*.x+.a32*.y+.a33*.z)
```

⟨*constant* Tachy_space.pi 474d⟩≡ (474i)
```
  let pi = 3.1415926
```

⟨*constant* Tachy_space.log10 474e⟩≡ (474i)
```
  let log10 = log 10.0
```

⟨*type* Tachy_space.ball 474f⟩≡ (474i)
```
  type ball = {
    tag : Tk.tagOrId;
    mutable x : float;
    mutable y : float;
    mutable z : float
   }
```

⟨*function* Tachy_space.create_tachy 474g⟩≡ (474i)
```
  let create_tachy top =
    let o = new space_tachy top in
    o#start;
```

⟨*toplevel* Tachy_space._1 474h⟩≡ (474i)
```
  let _ =
    let top = Applets.get_toplevel_widget [] in
    Wm.withdraw top;
    begin match Frx_dialog.f top (Mstring.gensym "foo")
          "Tachy test" "Use the space tachymeter"
      (Tk.Predefined "question") 1 ["Yes"; "No"] with
      0 -> Mmm.set_tachy create_tachy
    | _ -> ()
    end;
    destroy top
```

⟨extensions/tachy_space.ml 474i⟩≡
```
  open Safe418mmm

  open Tk

  module Provide = struct
    let capabilities = Capabilities.get()
    end

  module Mmm = Get(Provide)


  ⟨constant Tachy_space.mpoly_data 473⟩

  ⟨type Tachy_space.vector 474a⟩
  ⟨type Tachy_space.matrix 474b⟩

  let x_rotation a = let c = cos a and s = sin a in
      ((1.0,0.0,0.0), (0.0,c,s), (0.0,-.s,c))
```

```
and y_rotation a = let c = cos a and s = sin a in
    ((c,0.0,-.s), (0.0,1.0,0.0), (s,0.0,c))

⟨function Tachy_space.matrix_vector 474c⟩

⟨constant Tachy_space.pi 474d⟩
⟨constant Tachy_space.log10 474e⟩

⟨type Tachy_space.ball 474f⟩

class space_tachy (top : Widget.widget) =
 object (self)
  (* val top = top *)
  val mutable fr = top (* dummy initialisation *)
  val mutable c = top (* dummy initialisation *)
  val mutable c2 = top (* dummy initialisation *)
  val mutable mpoly = Tag "none" (* dummy initialisation *)
  val mutable i_park = Tag "none" (* dummy initialisation *)
  val mutable kilos = Tag "none" (* dummy initialisation *)
  val mutable pendings = Tag "none" (* dummy initialisation *)
  val mutable alive = false

  val mutable balls = Array.create 32 {tag = Tag "none"; x = 0.; y = 0.; z = 0.}
  val spacewidth = 4.0

  (* this one is private *)
  method start =
    fr <- Frame.create_named top "tachymeter" [BorderWidth (Pixels 2)];
    c <- Canvas.create fr
       [ Width (Pixels 72); Height (Pixels 72);
         BorderWidth (Pixels 1);
      Relief Sunken;
         HighlightThickness (Pixels 0);
         TakeFocus true (* pl3 fix *);
      Background Black];
    c2 <- Canvas.create fr [Width (Pixels 72); Height (Pixels 16)];
    pack [c; c2] [Side Side_Top; Fill Fill_X];

    i_park <-
       Canvas.create_rectangle c2
      (Pixels 1) (Pixels 1)
      (Pixels 4) (Pixels 4) [FillColor Black];

    kilos <-
       Canvas.create_text c2
        (Pixels 36) (Pixels 8)
     [Text "0"; Font "variable"];

    pendings <-
       Canvas.create_text c2
        (Pixels 68) (Pixels 8)
     [Text "0"; Font "variable"];

    balls <-
       Array.map (fun _ ->
     { tag = Canvas.create_line c [Pixels 100; Pixels 100;
                                   Pixels 100; Pixels 100]
                                   [FillColor (NamedColor "White")];
       x = Random.float spacewidth -. (spacewidth /. 2.0);
       y = Random.float spacewidth -. (spacewidth /. 2.0);
     z = Random.float 0.9 +. 0.1 }) balls;

  mpoly <- Canvas.create_polygon c (List.fold_right (fun (x,y) s ->
    let x = truncate ((x -. 26.0) *. 1.3) + 36
    and y = 36 - truncate ((20.6 +. y) *. 1.3)
    in [Pixels x; Pixels y] @ s)
                    mpoly_data [])
     [Width (Pixels 2); FillColor Green; Outline White];

  for i = 0 to Array.length balls - 1 do
    self#ball_update balls.(i) 0.0
  done;

  Canvas.lower_bot c2 pendings;

  bind c [[], Destroy] (BindSet ([], (fun _ -> alive <- false)));
  (* These bindings are specific to the applet version *)
  bind c [[], ButtonPressDetail 1]
    (BindSet ([], (fun _ -> Mmm.new_window_initial (); ())));
  bind c [[], ButtonPressDetail 2]
    (BindSet ([], (fun _ -> Mmm.new_window_sel (); ())));

  alive <- true;
  pack [fr][]

val mutable mx = 0.
val mutable my = 0.
val mutable last_speed = 0.
val mutable last_speed2 = 0.
val mutable last_total = 0
val mutable idle = false

method ball_update ball speed =
  let x = truncate (ball.x *. (0.2 /. ball.z) *. 32.0) + 36
  and y = truncate (ball.y *. (0.2 /. ball.z) *. 32.0) + 36
  and x' = truncate (ball.x *. (0.2 /. (ball.z +. speed)) *. 32.0) + 36
  and y' = truncate (ball.y *. (0.2 /. (ball.z +. speed)) *. 32.0) + 36
  in
  let x', y' = if (x,y) = (x',y') then x', y'+1 else x', y' in
  Canvas.coords_set c ball.tag
    [Pixels x; Pixels y; Pixels x'; Pixels y'];
  let x =
    Printf.sprintf "%02X" (truncate ((1.0 -. ball.z) /. (1.0 -. 0.1) *. 255.0)) in
  Canvas.configure_line c ball.tag [FillColor (NamedColor ("#"^x^x^x))]


method update speed total =
   Canvas.coords_set c mpoly
     (List.fold_right (fun (x,y) s ->
   let x = (x -. 26.0) *. 1.3
   and y = (20.6 +. y) *. 1.3
       in
   let (x,y,z) = matrix_vector (x_rotation mx) (matrix_vector (y_rotation my) (x,y,0.0)) in
     let r = (z +. 200.0) /. 200.0 in
     let x = truncate ( x *. r ) + 36
     and y = - truncate ( y *. r ) + 36
       in [Pixels x; Pixels y] @ s) mpoly_data []) ;
      mx <- mx +. 0.01;
      my <- my +. 0.02;
    if speed = 0.0 then begin
  if not idle then begin
    Canvas.configure_rectangle c2 i_park [FillColor Black;
                          Outline Black];
    idle <- true
  end
     end
    else begin
  Canvas.configure_rectangle c2 i_park [FillColor Green;
                        Outline Green];
      idle <- false
    end;
     if total <> last_total then
        Canvas.configure_text c2 kilos
       [Text (if total > 1000000 then
                Printf.sprintf "%d.%02dM" (total/1000000)
                                        ((total mod 1000000)/10000)
              else if total > 1000 then
            Printf.sprintf "%d.%01dK" (total/1000)
                                          ((total mod 1000)/100)
          else string_of_int total)];
     last_total <- total;
     let speed = if speed = 0. then 0. else log speed in
     (* Smooth *)
     last_speed2 <-
    if last_speed2 > speed *. 0.8 +. last_speed2 *. 0.2
    then last_speed2 -. 0.1
    else speed;
     let speeds =
   if last_speed2 -. last_speed > 0.2 then last_speed +. 0.2
   else if last_speed2 -. last_speed < (-0.1) then last_speed -. 0.1
   else last_speed
     in
     (* let speeds = last_speed *. 0.5 +. speed *. 0.5 in *)
     if abs_float (speeds -. last_speed) > 0.05 then begin
       last_speed <- speeds;
   let v = speeds /. log10 *. 0.02 in

   for i = 0 to Array.length balls - 1 do
     balls.(i).z <- balls.(i).z -. v;
     if( balls.(i).z < 0.1 ) then begin
       balls.(i).z <- 1.0;
       balls.(i).x <- Random.float spacewidth -. (spacewidth /. 2.0);
       balls.(i).y <- Random.float spacewidth -. (spacewidth /. 2.0)
     end;
     self#ball_update balls.(i) v;
   done;
       update_idletasks()
      end


method report_cnx n =
  if Winfo.exists c2 then
    if n = 0 then begin
  Canvas.configure_text c2 pendings [Text ""];
     Canvas.lower_bot c2 pendings
    end
    else begin
  Canvas.configure_text c2 pendings
    [Text (string_of_int n)];
     Canvas.raise_top c2 pendings
```

```
      end

  method report_busy busy =
    if Winfo.exists c2 then
      if busy then begin
       Canvas.lower_bot c2 pendings;
    Canvas.configure_rectangle c2 i_park [FillColor Red;
                          Outline Red];
    update_idletasks()
      end
      else begin
       Canvas.raise_top c2 pendings;
    Canvas.configure_rectangle c2 i_park [FillColor Black;
                          Outline Black]
      end

  method report_traffic tick_duration bytes_read sample_read =
    if alive then
      self#update (float sample_read *. 1000. /. float tick_duration)
    bytes_read

  method quit =
    alive <- false;
    destroy fr

end

⟨function Tachy_space.create_tachy 474g⟩
  (o :> Mmm.tachymeter)

⟨toplevel Tachy_space._1 474h⟩
```

F.11.6 extensions/tachy_test.ml

⟨*constant* Tachy_test.tachy_data 478⟩≡ (479f)

```
  (* inside bitmap, circle is in +16+7 +66+57, radius 25 *)

  let tachy_data = "GIF\056\057aP\000A\000\227\000\000\000\000\000\
  \044\044\044\060\000\000YYY\138\138\138\154\154\154\170\
  \170\170\186\186\186\203\203\203\219\219\219\231qq\235\235\
  \235\243yy\255\255\255\000\000\000\000\000\000\033\249\004\
  \001\000\000\009\000\044\000\000\000\000P\000A\000\000\004\
  \255\048\201I\171\189\024H\157\056\199\096\040\142\164\213\
  Hg\146\166e\235\190p\060\013\052\045\223DN\000\
  M\225\223\148\001\133\005\020\233\142\187\158oY\184\009\
  \039\196\162\005I\229\049\153\006\131E\032z\162\164\023\
  \042\178\144\188\250\178\232J\180\226U\021\063\146\129\248\
  \136\174c\235\233\009\220\212\232\175a\041\052sx\132\
  \133\133P\096\033p\130H\134\142\143\006\007z\137\032\
  \129rt\144\153x\007\156\136\148\147\051\151\058\154\164\
  Y\156\156\004\160\159n\009\003\000\152\165\164\167I\158\
  \171\173r\175\004\177\187\007I\157\018\092\171\140\059\187\
  \197\060\167\146\095\148\195\186\197\197\200\008\008\182\173\176\
  \206\188\007\209\210\194\213\214\187\217\209\171\141\221\206\223\
  \218\137\226\227\228\223\148\220\233\222\217\231\237\238\165\229\
  \096\242\143\001\001\003\001\006\174\053\006\249\000\210\131\039\
  \229\158\035\125\253\250\001\232\055\064\033\141\129\230\054\196\
  \064G\010\192\190\044\251\030fA\056\240\194\031\035\163\
  \255b\185J\008\208\031\198\128\177\216\133\044\165\049\203\
  \194\135\015\045\166\180\176\167\132AC\045\255\213\216\185\
  \203\227\011\138\243\172\197\203\017t\030\205\018\031n\022\
  \037u\164\150\136\020J\151f\002\064\181f\139\170\011\
  \221\037\152\231\039\133\213\011\031\186J\029\167\160\172\002\
  \167\033\160\046\025\219\205\236Y\025w\216\022s\043\241\
  \197\021\185\187\220\190e\229\034\046\094Mt\129\248\253\
  \011\041p\135\024\131\009\059\050\204\183\197\093\197\143\220\
  \050\144\178\022r\228\178\147\041\023\176\092X\001\131\204\
  E\054s\094\236\249s\162\209\139\063\131\150\130\186P\
  i\211\148Z\215y\189\026\140l\052\170a\127\186M\
  \123Z\029N\144\049\171\158\182\053R\175\172x\001\044\
  \016N\092\002\170cx\023\044P\094\182\249\004\095\192\
  \199J\151\174\220\250\016d\217\231m\031\239\253\176s\
  \240\008\196\143\151\190\225k\034\022\208\178u\091\191\125\
  \210G\091\156\202\201\039E\159\252\144\251\211\232\183N\
  \045qxH\208\095\125\229\149\032\096\005\180Lp\096\
  \130\048\012HA\131\009\244\007a\056\060\024\232\223\133\
  \024\158\192\030\135\032\134\040\226\136\036\194\016\001\000\059\
  "
```

⟨*constant* `Tachy_test.park_data` 479a⟩≡ (479f)

```
  let park_data =
  "#define break_width 15
  #define break_height 11
  static char break_bits[] = {
     0x0c, 0x18, 0xf4, 0x17, 0x3a, 0x2e, 0xba, 0x2d, 0xb9, 0x4d, 0x3d, 0x5e,
     0xb9, 0x4f, 0xba, 0x2f, 0xba, 0x2f, 0xf4, 0x17, 0x08, 0x08};
  "
```

⟨*constant* `Tachy_test.pi` 479b⟩≡ (479f)

```
  let pi = 3.1415926
```

⟨*constant* `Tachy_test.log10` 479c⟩≡ (479f)

```
  let log10 = log 10.0
```

⟨*function* `Tachy_test.create_tachy` 479d⟩≡ (479f)

```
  let create_tachy top =
    let o = new default_tachy top in
    o#start;
```

⟨*toplevel* `Tachy_test._1` 479e⟩≡ (479f)

```
  let _ =
    let top = Applets.get_toplevel_widget [] in
    Wm.withdraw top;
    begin match Frx_dialog.f top (Mstring.gensym "foo")
          "Tachy test" "Use this test tachymeter"
      (Tk.Predefined "question") 1 ["Yes"; "No"] with
      0 -> Mmm.set_tachy create_tachy
    | _ -> ()
    end;
    destroy top
```

⟨`extensions/tachy_test.ml` 479f⟩≡

```
  open Safe418mmm

  open Tk

  module Provide = struct
    let capabilities = Capabilities.get()
    end

  module Mmm = Get(Provide)

  (* Tachymeter *)

  (* gif is 80x65 *)
  ⟨constant Tachy_test.tachy_data 478⟩
  ⟨constant Tachy_test.park_data 479a⟩


  ⟨constant Tachy_test.pi 479b⟩
  ⟨constant Tachy_test.log10 479c⟩

  class default_tachy (top : Widget.widget) =
   object (self)
    (* val top = top *)
    val mutable canvas = top (* dummy initialisation *)
    val mutable alive = false

    (* Various components of the canvas, all with dummy init values *)
    val mutable i_park = Tag "none"
    val mutable kilos = Tag "none"
    val mutable aig = Tag "none"
    val mutable pendings = Tag "none"


    (* this one is private *)
    method start =
      let c =
        Canvas.create_named top "tachymeter"
         [Width (Pixels 80); Height (Pixels 80);
           BorderWidth (Pixels 0);
           HighlightThickness (Pixels 0);
           TakeFocus true (* pl3 fix *)] in
      (* Use colors so that images are not transparent *)
      (*
      let tachy_image =
        begin
         try
        let bgc = Tk.cget c CBackground in
        Protocol.tkEval
          [|Protocol.TkToken "set";
            Protocol.TkToken "TRANSPARENT_GIF_COLOR";
            Protocol.TkToken bgc |]; ()
         with _ -> ()
        end;
        *)
      let tachy_image = Frx_misc.create_photo [Data tachy_data]
      and park_image =
        Imagebitmap.create [Data park_data; Foreground Red] in

      i_park <-
        Canvas.create_rectangle c
       (Pixels 72) (Pixels 3)
       (Pixels 75) (Pixels 6) [FillColor Black];

      kilos <-
        Canvas.create_text c (Pixels 40) (Pixels 73) [Text "0"];
      aig <-
        Canvas.create_line c [Pixels 41; Pixels 32; Pixels 41; Pixels 57]
                         [Width (Pixels 2)];
      pendings <-
        Canvas.create_text c (Pixels 70) (Pixels 60) [Text "0"];

      let i_tachy =
        Canvas.create_image c (Pixels 0) (Pixels 0)
         [ImagePhoto tachy_image; Anchor NW]

      in

      Canvas.lower_bot c pendings;

      (* All other items must be put above the background image *)
      List.iter (fun i -> Canvas.raise_above c i i_tachy)
        [kilos; aig; i_park];

      bind c [[], Destroy] (BindSet ([], (fun _ -> alive <- false)));

      (* These bindings are specific to the applet version *)
      bind c [[], ButtonPressDetail 1]
        (BindSet ([], (fun _ -> Mmm.new_window_initial ())));
      bind c [[], ButtonPressDetail 2]
        (BindSet ([], (fun _ -> Mmm.new_window_sel ())));

      alive <- true;
      pack [c][];
      canvas <- c

    val mutable last_speed = 0.
    val mutable last_total = 0
    val mutable idle = false

    method update speed total =
      if speed = 0.0 then begin
        if not idle then begin
      Canvas.configure_rectangle canvas i_park [FillColor Black;
                            Outline Black];
      idle <- true
        end
      end
      else begin
        Canvas.configure_rectangle canvas i_park [FillColor Green;
                           Outline Green];
        idle <- false
      end;
      if total <> last_total then
        Canvas.configure_text canvas kilos [Text (string_of_int total)];
      last_total <- total;
      let speed = if speed = 0. then 0. else log speed in
         (* Smooth *)
      let speeds = (last_speed +. speed) /. 2. in
      if abs_float (speeds -. last_speed) > 0.1 then begin
        last_speed <- speeds;
        let v = speeds /. log10 in
        let angle = v /. 4.0 *. pi in
        let angle = if angle < 0.1 then 0.0 else angle in
        let x = 41.0 -. (sin angle *. 25.0)
        and y = 32.0 +. (cos angle *. 25.0) in
```

```
        Canvas.coords_set canvas aig
      [Pixels 41; Pixels 32;
        Pixels (truncate x); Pixels (truncate y)];
        update_idletasks()
      end

    method report_cnx n =
      if Winfo.exists canvas then
        if n = 0 then begin
      Canvas.configure_text canvas pendings [Text ""];
         Canvas.lower_bot canvas pendings
        end
        else begin
      Canvas.configure_text canvas pendings
        [Text (string_of_int n)];
         Canvas.raise_top canvas pendings
        end

    method report_busy busy =
      if Winfo.exists canvas then
        if busy then begin
         Canvas.lower_bot canvas pendings;
      Canvas.configure_rectangle canvas i_park [FillColor Red;
                                Outline Red];
      update_idletasks()
        end
        else begin
         Canvas.raise_top canvas pendings;
      Canvas.configure_rectangle canvas i_park [FillColor Black;
                                Outline Black]
        end

    method report_traffic tick_duration bytes_read sample_read =
      if alive then
        self#update (float sample_read *. 1000. /. float tick_duration)
      bytes_read

    method quit =
      alive <- false;
      destroy canvas

  end

  ⟨function Tachy_test.create_tachy 479d⟩
    (o :> Mmm.tachymeter)

  ⟨toplevel Tachy_test._1 479e⟩
```

F.12 applets/

applets.mli

⟨applets.mli 482⟩≡

```
  ⟨type Applets.applet_callback 176f⟩

  ⟨signature Applets.register 184a⟩
```

⟨*signature* Applets.error 180c⟩

⟨*signature* Applets.call 178e⟩

⟨*signature* Applets.get_toplevel_widget 188e⟩

applets.ml

⟨applets.ml 483a⟩≡

⟨*copyright header calves* 14b⟩

⟨*type* Applets.applet_callback 176f⟩

⟨*function* Applets.register 185a⟩

⟨*function* Applets.error 180d⟩

⟨*function* Applets.call 186a⟩

⟨*function* Applets.get_toplevel_widget 188f⟩

appsys.ml

⟨appsys.ml 483b⟩≡

```
(* This module of the applet system is specific to MMM *)
```

⟨*constant* Appsys.active 177f⟩

⟨*constant* Appsys.types 177h⟩

⟨*function* Appsys.activate 177i⟩
⟨*function* Appsys.deactivate 177j⟩

⟨*function* Appsys.pref_init 177e⟩
⟨*function* Appsys.pref_set 177g⟩

⟨*function* Appsys.applets_pref 177d⟩

⟨*function* Appsys.load_initial_modules 188j⟩

⟨*function* Appsys.init 177b⟩

appsys.mli

⟨appsys.mli 483c⟩≡

⟨*signature* Appsys.init 177a⟩

appview.ml

⟨appview.ml 483d⟩≡

```
(* was in viewers.mli
class trivial_display : (Widget.widget * Url.t) -> (* #display_info *)
(* boilerplate class type *)
object
```

```
    method di_abort : unit
    method di_destroy : unit
    method di_fragment : string option -> unit
    method di_last_used : int
    method di_load_images : unit
    method di_redisplay : unit
    method di_source : unit
    method di_title : string
    method di_touch : unit
    method di_widget : Widget.widget
    method di_update : unit
  end
  *)

  ⟨class Appview.trivial_display 178f⟩

  (* Wrapping up
   * When EMBED is recognized by the HTML display machine, the "embedded object"
   * is passed to the embed manager/scheduler with some viewer as continuation.
   * This module defines this viewer, which takes as arguments
   *   [parms] MIME parameters
   *   [frame] embedded frame in HTML widget
   *   [ctx] Viewers.context
   *   [doc] : definition of document
   * When the viewer is called, the bytecode may or may not have been loaded
   * (the embed manager only stores the applet in a file, it doesn't load it;
   *  the first invocation of the applet has the responsability to actually
   *  load the file)
   *)

  ⟨function Appview.is_update 179c⟩

  ⟨function Appview.applet_viewer 179f⟩

  ⟨function Appview.code_viewer 178a⟩
```

appview.mli

```
⟨appview.mli 484a⟩≡

  ⟨signature Appview.code_viewer 176a⟩
  ⟨signature Appview.applet_viewer 176b⟩
```

capabilities.ml

```
⟨exception Capabilities.Denied 484b⟩≡                                   (492d 489c)
  exception Denied (* access denied *)

⟨type Capabilities.mode 484c⟩≡                                          (492d 489c)
  type mode = Fixed | Extend | Temporary (* extension mode for access rights *)

⟨function Capabilities.string_of_mode 484d⟩≡                                 (489c)
  let string_of_mode = function
      Fixed -> ""
    | Extend -> I18n.sprintf "repeated"
    | Temporary -> I18n.sprintf "temporary"
```

⟨*type* Capabilities.right 485a⟩≡ (492d 489c)
```
  (* The rights as stored *)
  type right =
     FileR of string   (* read access to files *)
   | FileW of string   (* write acces to files *)
   | DocumentR of string   (* read access to URLs *)
      (* Document read access affects decoders, embedded viewers, as well as
         the general retrieval mechanism. It means that the applet has
         access to the document body. (Navigation, that is triggering
         retrieval/display of documents is always available, since inspection
         of URL is essentially useless).
       *)
   | HTMLDisplay
     (* HTML display machine access : this is very liberal; if granted, it
        means that the applet has access to *all* retrieved HTML documents
      *)
   | Internals
```

⟨*type* Capabilities.t 485b⟩≡ (492d 489c)
```
  type t = {
    mutable mode : mode;
    mutable rights : Rights.t;
    who: Url.t; (* where this applet was loaded from. *)
    }
```

⟨*function* Capabilities.local_default 485c⟩≡ (489c)
```
  (* For applets loaded from disk, we basically authorize access to any
     HTML document and browser extensions
    *)
  let local_default url = {
    mode = Extend;
    rights =
      List.fold_right Rights.add
        [true, HTMLDisplay;
         true, DocumentR ".*"]
        Rights.empty;
    who = url;
    }
```

⟨*function* Capabilities.lenient_default 485d⟩≡ (489c)
```
  (* For signed applets, we start from an empty set of rights, but allow
     right extension requests *)
  let lenient_default url = {
    mode = Extend;
    rights = Rights.empty;
    who = url;
    }
```

⟨*function* Capabilities.strict_default 485e⟩≡ (489c)
```
  (* For unsigned applets, but then, these are inherently unsafe,
     so there's no point defining any policy *)
  let strict_default url = {
    mode = Temporary;
    rights = Rights.empty;
    who = url;
    }
```

⟨*constant* Capabilities.current_capa 485f⟩≡ (489c)
```
  (* This is the crucial reference from which an applet should get its
     access rights at load-time (and not later) *)
  let current_capa = ref None
```

⟨*function* Capabilities.set 486a⟩≡ (489c)
```
  (* Call this to init to some value BEFORE loading some bytecode *)
  let set h = current_capa := Some h
```

⟨*function* Capabilities.reset 486b⟩≡ (489c)
```
  (* Call this AFTER loading the bytecode. *)
  let reset () = current_capa := None
```

⟨*function* Capabilities.check_FileR 486c⟩≡ (489c)
```
  (* For file names, we must of course remove dots before doing checks.
   * Also, we must be aware that a control context-switch between the
   * return of this function and the use of its result may be the occasion
   * for the applet to physically change the filename string. This race
   * condition may lead to security breach.
   * The only proper usage is to first make a copy of the string, and then
   * check the rights on the copy and use the copy if access is granted.
   * NOTE: even with this precaution, we still have the infamous Unix access()
   * race condition (although here it's probably more difficult to exploit).
   *)

  let check_FileR capa s =
    let s = Lexpath.remove_dots s in
    try
      Rights.iter (function
        | true, FileR r when Str.string_match (Str.regexp r) s 0 ->
        failwith "yes"
        | false, FileR r when r = s ->
        failwith "yes"
        | _ -> ())
       capa.rights;
      false
    with
      Failure "yes" -> true
    | _ -> false (* be conservative... *)
```

⟨*function* Capabilities.check_FileW 486d⟩≡ (489c)
```
  let check_FileW capa s =
    let s = Lexpath.remove_dots s in
    try
      Rights.iter (function
         true, FileW r when Str.string_match (Str.regexp r) s 0 ->
        failwith "yes"
        | false, FileW r when r = s ->
        failwith "yes"
        | _ -> ())
       capa.rights;
      false
    with
      Failure "yes" -> true
    | _ -> false (* be conservative... *)
```

⟨*function* Capabilities.check_DocumentR 486e⟩≡ (489c)
```
  (* Do we need to remove ../ here ? *)
  let check_DocumentR capa s =
    try
      Rights.iter (function
         true, DocumentR r when Str.string_match (Str.regexp r) s 0 ->
        failwith "yes"
        | false, DocumentR r when r = s ->
        failwith "yes"
        | _ -> ())
       capa.rights;
      false
    with
      Failure "yes" -> true
    | _ -> false (* be conservative... *)
```

⟨*function* `Capabilities.check_HTMLDisplay` 487a⟩≡ (489c)

```
 let check_HTMLDisplay capa _ =
  try
    Rights.iter (function
       _, HTMLDisplay -> failwith "yes"
      | _ -> ())
     capa.rights;
    false
  with
    Failure "yes" -> true
  | _ -> false (* be conservative... *)
```

⟨*function* `Capabilities.check_Internals` 487b⟩≡ (489c)

```
 let check_Internals capa _ =
  try
    Rights.iter (function
       _, Internals -> failwith "yes"
      | _ -> ())
     capa.rights;
    false
  with
    Failure "yes" -> true
  | _ -> false (* be conservative... *)
```

⟨*type* `Capabilities.question` 487c⟩≡ (489c)

```
 (* Ask for a capability
  *  isregexp is true when the applet requires caps during load-time
  *   we then simply popup the question
  *  otherwise we check if it's been granted or ask the user (unless
  *  mode is Fixed)
  *)
 type 'a question = {
   check_right: t -> string -> bool;
   make_right : string -> right;
   question_simple: (string -> string -> 'a, unit, string) format;
   question_regexp: (string -> string -> 'a, unit, string) format
   }
```

⟨*type* `Capabilities.cright` 487d⟩≡ (489c)

```
 type cright =
   CFileR | CFileW | CDocumentR | CHTMLDisplay | CInternals
```

⟨*constant* `Capabilities.table` 487e⟩≡ (489c)

```
 (* The argument passed to make_right must be a value owned by US
     (that is, it must not be mutated by the applet)
  *)
 let table = [
   CFileR,
   { check_right = check_FileR;
     make_right = (fun s -> FileR s);
     question_simple = "Grant %s read access to the file\n%s";
     question_regexp = "Grant %s read access to files matching\n%s"};
   CFileW,
   { check_right = check_FileW;
     make_right = (fun s -> FileW s);
     question_simple = "Grant %s write access to the file\n%s";
     question_regexp = "Grant %s write access to files matching\n%s"};
   CDocumentR,
   { check_right = check_DocumentR;
     make_right = (fun s -> DocumentR s);
     question_simple = "Grant %s read access to document\n%s";
     question_regexp = "Grant %s read access to documents matching\n%s"};
   CHTMLDisplay,
   { check_right = check_HTMLDisplay;
     make_right = (fun _ -> HTMLDisplay);
     question_simple = "Grant %s access to HTML display machine\n%s";
     question_regexp = "Grant %s access to HTML display machine\n%s"};
   CInternals,
   { check_right = check_Internals;
     make_right = (fun _ -> Internals);
     question_simple = "Grant %s access to MMM internals\n%s";
     question_regexp = "Grant %s access to MMM internals\n%s";}
 ]
```

⟨*function* Capabilities.get_question 488a⟩≡ (489c)

```
  let get_question = function
    | FileR s -> s, List.assoc CFileR table
    | FileW s -> s, List.assoc CFileW table
    | DocumentR s -> s, List.assoc CDocumentR table
    | HTMLDisplay -> "", List.assoc CHTMLDisplay table
    | Internals -> "", List.assoc CInternals table
```

⟨*function* Capabilities.ask 488b⟩≡ (489c)

```
  (* This is the function available for Safe libraries *)
  (* REMEMBER TO MAKE COPIES OF ARGUMENT IF MUTABLE *)
  let ask capa r =
    let param, q = get_question r
    and mode = string_of_mode capa.mode in
    if q.check_right capa param then true (* already granted *)
    else begin
      let title =
        I18n.sprintf "Security check for %s" (Url.string_of capa.who) in
      let question = I18n.sprintf q.question_simple mode param in
      let granted = Error.choose (I18n.sprintf "%s\n%s\n" title question) in
      if granted && capa.mode = Extend then
        capa.rights <- Rights.add (false, q.make_right param) capa.rights;
      granted
    end
```

⟨*function* Capabilities.require_capa 488c⟩≡ (489c)

```
  (* Here, we make copies ourselves *)
  let require_capa capa r =
    let param, q = get_question r
    and mode = string_of_mode capa.mode in
    (* old: let param = String.copy param in, but string are immutable now *)
    if capa.mode = Extend then begin
      let title =
        I18n.sprintf "Security Rights asked by %s" (Url.string_of capa.who) in
      let question = I18n.sprintf q.question_regexp mode param in
      let granted = Error.choose (I18n.sprintf "%s\n%s\n" title question) in
      if granted then
        capa.rights <- Rights.add (false, q.make_right param) capa.rights;
      granted
    end
    else (* not authorized to extend rights. Only "ask" will work. *)
      false
```

⟨*function* Capabilities.get 489a⟩≡ (489c)

```
  (*
   * This is exported to applets
   *)
  let get () =
    match !current_capa with
      None -> raise Not_found
    | Some h -> h
```

⟨*function* Capabilities.require 489b⟩≡ (489c)

```
  let require capa l =
    List.fold_right (&&)
      (List.map (require_capa capa) l)
      true
```

⟨capabilities.ml 489c⟩≡

```
  ⟨copyright header calves 14b⟩

  (*
   A simple capability manager : each applet get its own list of access rights.

   The capabilities are given *at load-time* for a bytecode, using the
   current_capa reference and its accessorts get/set/reset. current_capa
   is non-empty only during load-time. An attempt to "get" after load-time
   will result in a Not_found exception.
   (Dynamic loading must be in a critical section anyway, since Dynlink
    would not support interleaved loading. The only reason for which
    a "context switch" would occur is when there is toplevel expression
    in the applet that causes Tk to enter its even loop (remember we
    don't have true threads here).)

   NOTE: about a possible race condition (an active applet calling
         Capabilities.get() during its execution, while we are
         loading another applet and current_capa is Some c)

    this race condition may happen only if some toplevel expression
    in the applet being loaded yields to the Tk event loop (because
    otherwise the loading is completely synchronous).

    Suppose this happens.

    The active applet now has a handle to the capabilities (type t)
    that were granted to the applet being loaded. The only possible
    operation is to require more rights for this new applet. The
    request will correctly appear to the user.

    But since capabilites are stored in the closures of dangerous
    functions at load-time (by functor application), there is no
    way for the active applet to benefit from the rights of the
    applet being loaded.
    (assuming that loading of applets can't be interleaved, which is
     true by virtue of "in_load")

   NOTE: about checking rights
    When checking the rights on a given "ressource description" (e.g.
    a filename or an URL), we must be sure that this ressource cannot
    be modified by the applet between the moment we check it and the
    moment we actually use it.

  *)

  ⟨exception Capabilities.Denied 484b⟩

  ⟨type Capabilities.mode 484c⟩

  (* Fixed means that the only rights are the initial rights.
     Extend means that a right may be requested, and if granted,
       will be granted for any further requests.
     Temporary means that a right may be granted temporarily, but must be
       granted again if requested for later.
   *)

  ⟨function Capabilities.string_of_mode 484d⟩

  (* Since we don't have subtyping of signatures, there is no easy way
     to make access rights extensible. Even if we export separate functions
     for requesting each kind of access right, we still get different
     signatures. Thus, it doesn't make any difference for the applet author
     if we export this type or if we export construction functions.
     For example, HTMLDisplay is not available for Calves applets, but
     we do export it. In the future, we would have to use the Safe$(VERSION)
     mechanism to ensure backward compatibility...
   *)
  ⟨type Capabilities.right 485a⟩

  module Rights = Set.Make(struct type t = bool * right
                                 let compare = compare end)
    (* the flag indicates that regexp matching should be used *)


  (* The information that will be put in the argument structure of functors
     producing modules when access control is needed.
     This is the easiest way to stick this value in lots of closures at the
     same time, as well as keeping signatures of modules identical to their
     original version (without access control).
     IMPORTANT NOTE:  the "who" field, an Url.t, is mutable (it would also
     be mutable in string form).
     Hence, we want to make a copy of it if we want to give it to the applet,
     and keep our own version private so that the applet can't change it under
     our feet...
     *)

  ⟨type Capabilities.t 485b⟩

  (*
   * Various constructors
   *)

  ⟨function Capabilities.local_default 485c⟩

  ⟨function Capabilities.lenient_default 485d⟩

  ⟨function Capabilities.strict_default 485e⟩


  ⟨constant Capabilities.current_capa 485f⟩
```

```
  ⟨function Capabilities.set 486a⟩
  ⟨function Capabilities.reset 486b⟩


  (*
   * Various checks
   *)

  ⟨function Capabilities.check_FileR 486c⟩

  ⟨function Capabilities.check_FileW 486d⟩

  ⟨function Capabilities.check_DocumentR 486e⟩

  ⟨function Capabilities.check_HTMLDisplay 487a⟩

  ⟨function Capabilities.check_Internals 487b⟩


  (* GUI *)
  open Tk

  ⟨type Capabilities.question 487c⟩


  ⟨type Capabilities.cright 487d⟩

  ⟨constant Capabilities.table 487e⟩

  ⟨function Capabilities.get_question 488a⟩

  ⟨function Capabilities.ask 488b⟩

  ⟨function Capabilities.require_capa 488c⟩


  (* TODO: we would also need a "security editor" *)


  ⟨function Capabilities.get 489a⟩

  ⟨function Capabilities.require 489b⟩
```

capabilities.mli

```
⟨signature Capabilities.local_default 491a⟩≡                          (492d)
  val local_default : Url.t -> t

⟨signature Capabilities.lenient_default 491b⟩≡                        (492d)
  val lenient_default : Url.t -> t

⟨signature Capabilities.strict_default 491c⟩≡                         (492d)
  val strict_default : Url.t -> t

⟨signature Capabilities.set 491d⟩≡                                    (492d)
  val set : t -> unit

⟨signature Capabilities.reset 491e⟩≡                                  (492d)
  val reset : unit -> unit
```

⟨*signature* Capabilities.ask 492a⟩≡ (492d)
```
  val ask: t -> right -> bool
    (* [ask capa right] *)
    (* REMEMBER TO MAKE COPIES OF ARGUMENT IF MUTABLE (eg string) *)
```

⟨*signature* Capabilities.require 492b⟩≡ (492d)
```
  val require: t -> right list -> bool
    (* get some specific capabilities, to avoid popping dialog boxes all
       over the place. Moreover, can make use of regexp
     *)
```

⟨*signature* Capabilities.get 492c⟩≡ (492d)
```
  val get : unit -> t
    (* get the default capabilities *)
```

⟨capabilities.mli 492d⟩≡
```

  ⟨type Capabilities.right 485a⟩

  module Rights : Set.S with type elt = bool * right

  ⟨type Capabilities.mode 484c⟩

  ⟨type Capabilities.t 485b⟩

  ⟨signature Capabilities.local_default 491a⟩
  ⟨signature Capabilities.lenient_default 491b⟩
  ⟨signature Capabilities.strict_default 491c⟩


  ⟨signature Capabilities.set 491d⟩
  ⟨signature Capabilities.reset 491e⟩

  ⟨signature Capabilities.ask 492a⟩


  ⟨exception Capabilities.Denied 484b⟩

  ⟨signature Capabilities.require 492b⟩

  (* IMPORTANT: The following functions may only be called at load-time *)

  ⟨signature Capabilities.get 492c⟩
```

dload.ml

⟨*function* Dload.dynlinkerror 492e⟩≡ (493)
```
  (* This is now available as Dynlink.error_message, but not i18n *)
  let dynlinkerror = function
     Dynlink.Not_a_bytecode_file _s ->
       I18n.sprintf "Not a bytecode file"
   | Dynlink.Inconsistent_import s ->
       I18n.sprintf "Inconsistent import: %s" s
   | Dynlink.Unavailable_unit s ->
       I18n.sprintf "Unavailable unit: %s " s
   | Dynlink.Unsafe_file ->
       I18n.sprintf "Unsafe file"
   | Dynlink.Linking_error (s, Dynlink.Undefined_global v) ->
       I18n.sprintf "Error while linking: %s Undefined global: %s" s v
   | Dynlink.Linking_error (s, Dynlink.Unavailable_primitive v) ->
       I18n.sprintf "Error while linking: %s Unavailable primitive: %s" s v
   | Dynlink.Corrupted_interface _s ->
       I18n.sprintf "Corrupted interface"
   | Dynlink.Linking_error (s, Dynlink.Uninitialized_global v) ->
       I18n.sprintf "Error while linking: %s Uninitialized global: %s" s v
  (*
   | Dynlink.File_not_found s ->
       I18n.sprintf "Cannot find file %s in search path" s
   | Dynlink.Cannot_open_dll s ->
       I18n.sprintf "Error while loading shared library: %s" s
  *)
   | other ->
       Dynlink.error_message other
```

⟨dload.ml 493⟩≡

```
 ⟨copyright header calves 14b⟩

 open Common
 open Fpath_.Operators

 ⟨constant Dload.paranoid 187d⟩

 ⟨function Dload.dynlinkerror 492e⟩


 ⟨type Dload.applet_callback 176e⟩

 ⟨type Dload.t 176d⟩

 ⟨type Dload.mod_status 178c⟩

 ⟨constant Dload.mod_cache 180f⟩

 ⟨function Dload.get 181d⟩
 ⟨function Dload.iter 181b⟩
 ⟨function Dload.remove 181a⟩

 ⟨constant Dload.register_queue 184b⟩

 ⟨function Dload.register 185c⟩

 ⟨function Dload.register_flush 185d⟩

 (* We need to resynchronize applet evaluation : several <EMBED> may
    use the same SRC bytecode. The navigator will request us to load
    several times the same bytecode.
  *)

 ⟨constant Dload.pending_loads 185e⟩

 ⟨function Dload.add_pending_applet 185f⟩

 ⟨function Dload.flush_pending_applets 185g⟩


 (* Dynlink is not reentrant + security requires it to be in critical
    section anyway (to protect capabilities)
    This reference protects:
        1- Dynlink
        2- The queue of functions
  *)

 ⟨constant Dload.in_load 183e⟩

 ⟨function Dload.load_local 189b⟩

 ⟨function Dload.unsafe_load 184c⟩


 ⟨function Dload.ask 183c⟩


 (* In which form is the applet ?
  * We'd like to push a solution with a single MIME type
  *   application/x-caml-applet
  *   and an attribute specifying the encoding : encoding  = source/bytecode
  * However:
  *  1- it's difficult to set up MIME attributes on servers (at least Apache)
  *  2- a navigator might not pass this information to a plugin
  * Thus:
  *  in case the attribute is not present, we try to compute it from
  *  the URL suffix, and magic numbers.
  *)


 ⟨type Dload.applet_kind 182c⟩

 ⟨function Dload.applet_kind 183a⟩

 ⟨function Dload.load 182b⟩
```

dload.mli

```
⟨dload.mli 494⟩≡

  ⟨type Dload.applet_callback 176e⟩

  ⟨type Dload.t 176d⟩

  ⟨type Dload.mod_status 178c⟩

  ⟨signature Dload.get 178b⟩
  ⟨signature Dload.remove 179b⟩
  ⟨signature Dload.iter 181c⟩

  ⟨signature Dload.register 185b⟩
  ⟨signature Dload.add_pending_applet 181f⟩

  ⟨signature Dload.load 181g⟩

  ⟨signature Dload.load_local 189a⟩

  ⟨signature Dload.paranoid 187c⟩

  ⟨signature Dload.in_load 183d⟩
```

pgp.ml

```
⟨pgp.ml 495⟩≡
  ⟨copyright header calves 14b⟩
  (* A PGP decoder *)

  ⟨toplevel comment Pgp 190b⟩

  ⟨function Pgp.read_all 191b⟩

  ⟨function Pgp.batch_pgp 191a⟩

  ⟨function Pgp.check 190c⟩
```

Glossary

```
URL  = Uniform Resource Locator
URI  = Universal Resource Identifier
HTML =
DOM  = Document Object Model
CSS  = Cascading Style Sheets
JS   = Javascript
HTTP =
WWW  = World Wide Web
MIME = Multi-Purpose Internet Mail Extensions

DID  = Document IDentifier
WR   = Web Request
WWWR = Web Request
```

Indexes

Here is a list of the identifiers used, and where they appear. Underlined entries indicate the place of definition. This index is generated automatically.

Bibliography

[Knu92] Donald E. Knuth. *Literate Programming.* Center for the Study of Language and Information, 1992. cited page(s) 13

[Pad09] Yoann Padioleau. Syncweb, literate programming meets unison, 2009. https://github.com/aryx/syncweb. cited page(s) 13