Principia Softwarica: The Plan 9 Web Browser `mothra`

version 0.1

Yoann Padioleau
yoann.padioleau@gmail.com

with code from
Tom Duff, David Hogan, Russ Cox

April 22, 2026

Contents

1 Introduction **7**
- 1.1 Motivations . . . 7
- 1.2 The Plan 9 web browser: `mothra` . . . 8
- 1.3 Other web browsers . . . 8
- 1.4 Getting started . . . 9
- 1.5 Requirements . . . 9
- 1.6 About this document . . . 10
- 1.7 Copyright . . . 10
- 1.8 Acknowledgments . . . 10

2 Overview **11**
- 2.1 Web browser principles . . . 11
  - 2.1.1 A viewer for hypertext documents . . . 11
  - 2.1.2 The URL as universal name . . . 11
  - 2.1.3 Client/server over a text protocol . . . 12
  - 2.1.4 Rendering text with structure (no layout engine required) . . . 12
  - 2.1.5 Composition through pipes and file servers . . . 12
  - 2.1.6 Mothra for the pre-CSS, pre-JS web . . . 12
- 2.2 The Plan 9 web stack interfaces . . . 13
  - 2.2.1 `/mnt/web` (`webfs`) . . . 13
  - 2.2.2 `/mnt/webcookies` . . . 13
  - 2.2.3 Pipelines: `fetch | uhtml | render` . . . 13
  - 2.2.4 `/net/tls` and HTTPS . . . 13
- 2.3 `mothra` command-line interface . . . 13
- 2.4 `hello.html` . . . 14
- 2.5 Code organization . . . 14
- 2.6 Software architecture . . . 14
  - 2.6.1 Trace of a web request: `mothra http://example.com` . . . 14
- 2.7 Book structure . . . 15

3 Core Data Structures **17**
- 3.1 URLs . . . 17
- 3.2 HTTP requests and responses . . . 17
- 3.3 The `/mnt/web` interface . . . 17
- 3.4 Cookies (`Jar`, `Cookie`) . . . 17
- 3.5 Charset tables . . . 17
- 3.6 HTML parse tree (mothra's flat item list) . . . 17
- 3.7 Mothra's view (text runs, layout, panels) . . . 17

4 `mothra`: `main()` and Panels **18**
4.1 Command-line arguments . . . 18
4.2 Panel layout (address bar, display, status) . . . 18
4.3 The main I/O loop . . . 18
4.4 Reading from a URL or stdin . . . 18

I Retrieving the Page **19**

5 Resolving URLs **20**
5.1 Webfs's URL algebra (`webfs/url.c`, RFC 3986) . . . 20
5.1.1 Parsing absolute URLs (scheme, authority, path, query, fragment) . . . 20
5.1.2 Authority and userinfo . . . 20
5.1.3 Resolving relative URLs against a base . . . 20
5.1.4 Path normalization (dots) . . . 20
5.1.5 Percent-encoding . . . 20
5.1.6 Form-data encoding . . . 20
5.2 Mothra's URL adapter (`mothra/url.c`) . . . 20

6 An HTTP Client: `hget` **21**
6.1 `main()` . . . 21
6.2 Connecting (TCP and TLS) . . . 21
6.3 Building the request (`GET`, `HEAD`) . . . 21
6.4 Reading the response (status, headers, body) . . . 21
6.5 Following redirects . . . 21
6.6 Range requests and resume . . . 21

7 The Web File Server: `webfs` **22**
7.1 `main()` and the 9P loop . . . 22
7.2 The `clone`/`ctl`/`body` protocol . . . 22
7.3 Connection lifecycle . . . 22
7.4 HTTP transport (`http.c`) . . . 22
7.4.1 The request (GET, POST, HEAD) . . . 22
7.4.2 Status codes . . . 22
7.4.3 Headers (content-type, content-length, content-encoding) . . . 22
7.5 Authentication (basic, digest, `www-authenticate`) . . . 22
7.6 Redirect handling and loop detection . . . 22
7.7 Per-client thread dispatch (`client.c`) . . . 22
7.8 Async I/O primitives (`io.c`: `iotlsdial`, `ioprint`) . . . 22
7.9 HTTPS: `tlsClient` and `/net/tls` . . . 22
7.10 Why a file server for the network . . . 22

8 The Cookie File Server: `webcookies` **23**
8.1 `main()` . . . 23
8.2 Storage format . . . 23
8.3 Domain matching and expiry . . . 23
8.4 The dual cookie architecture: `webfs/cookies.c` vs standalone `webcookies` . . . 23
8.5 Why split from `webfs` . . . 23

9 Retrieval and Progress **24**
9.1 Opening `/mnt/web/clone` and writing the URL to `ctl` . . . 24
9.2 Reading status, headers, and body . . . 24
9.3 The retrieval state machine . . . 24
9.4 Progress tracking and status-bar updates . . . 24
9.5 Scheduling concurrent fetches (main document + images) . . . 24
9.6 Lifecycle: abort, refresh, redisplay . . . 24

10 Charset Conversion: `tcs` and `uhtml` **25**
10.1 Why charset conversion lives outside the browser . . . 25
10.2 `tcs` as a pipe-friendly iconv (interface, not internals) . . . 25
10.3 `uhtml`: sniffing and shelling out . . . 25
10.4 The full pipeline: `fetch | uhtml | render` . . . 25

II Rendering the Page **26**

11 HTML Parsing: `rdhtml.c` **27**
11.1 What mothra parses (HTML4-ish, lenient) . . . 27
11.2 Tokenization: tags, attributes, text runs . . . 27
11.3 The tag dispatch table . . . 27
11.4 Building a flat list of items (no DOM) . . . 27
11.5 Entities and character references . . . 27
11.6 Surviving real-world HTML . . . 27

12 Content Dispatch **28**
12.1 MIME sniffing (`snoop.c`) . . . 28
12.2 MIME detection via `/bin/file -m` subprocess . . . 28
12.3 The viewer table (HTML, PLAIN, JPEG, GIF, PNG, . . .) . . . 28
12.4 Fallback logic: `image/` and `text/` prefix rules . . . 28
12.5 Dispatching to the right reader . . . 28
12.6 Handling non-HTML content . . . 28

13 Laying Out the Page **29**
13.1 From parsed items to on-screen text runs . . . 29
13.2 Word wrapping and reflow . . . 29
13.3 Fonts and text styles . . . 29
13.4 Hyperlinks and their rendering . . . 29
13.5 Scrolling and repaint . . . 29

14 Tag-by-Tag Rendering **30**
14.1 Structural (`<html>`, `<head>`, `<body>`) . . . 30
14.2 Headings and paragraphs . . . 30
14.3 Text styles (`<b>`, `<i>`, `<tt>`, `<em>`, . . .) . . . 30
14.4 Lists (`<ul>`, `<ol>`, `<dl>`) . . . 30
14.5 Breaks and rules (`<br>`, `<hr>`) . . . 30
14.6 Anchors (`<a>`) . . . 30

15 Tables **31**
15.1 HTML tables: `<table>`, `<tr>`, `<td>` . . . 31
15.2 What mothra does: linear pass-through . . . 31
15.3 Why full table layout is hard (column widths, rowspan, colspan) . . . 31
15.4 Comparison with abaco and full browsers . . . 31

16 Images **32**
16.1 The `<img>` tag . . . 32
16.2 `getpix` as a subprocess image fetcher . . . 32
16.3 The three-stage pipeline: `snooptype()` → format filter → `resize` . . . 32
16.4 Process management (`rfork`, `waitpid`) . . . 32
16.5 `resize` for scaling . . . 32
16.6 Image formats (JPEG/GIF/PNG via `libdraw`) . . . 32
16.7 Lazy loading and placeholders . . . 32
16.8 Debug instrumentation (`-d` flag, `poolcheck`, heap dumps) . . . 32

17 Forms **33**
17.1 The `<form>` tag and `forms.c` . . . 33
17.2 The `Field` state machine (TYPEIN, CHECK, PASSWD, RADIO, . . .) . . . 33
17.3 Input widgets (`<input type=...>`) . . . 33
17.4 Text areas and select menus . . . 33
17.5 Password fields . . . 33
17.6 Form encoding . . . 33
17.6.1 `uencodeform` (application/x-www-form-urlencoded) . . . 33
17.6.2 `mencodeform` (multipart/form-data) . . . 33
17.7 Submission: GET vs POST . . . 33

III Beyond the Basics **34**

18 Navigation and History **35**
18.1 Following links . . . 35
18.2 Back, forward, and history . . . 35
18.3 Bookmarks and hotlist . . . 35

19 Events and Integration **36**
19.1 Mouse clicks and scrolling . . . 36
19.2 Keyboard shortcuts . . . 36
19.3 The `plumb(2)` protocol . . . 36
19.4 Plumb messages: MIME type + filename . . . 36
19.5 Spawning external viewers . . . 36
19.6 Snarf (selection and clipboard) . . . 36

20 Developer Tools **37**
20.1 View source . . . 37
20.2 HTTP tracing (`-d` flag) . . . 37
20.3 MIME-type dumping (`snoop.c`) . . . 37
20.4 The debug-instrumentation pattern (fork + `poolcheck` + temp files) . . . 37
20.5 Saving a page . . . 37

21 Advanced Topics **38**
21.1 Proxy (via `$http_proxy`) . . . 38
21.2 Referer header . . . 38
21.3 Content encoding (gzip) . . . 38
21.4 Animated GIFs . . . 38
21.5 File protocol (`file://`) . . . 38
21.6 FTP protocol support (`doftp` in hget, ftp in `webfs/url.c`) . . . 38
21.7 External viewers via plumbing . . . 38
21.8 Caching in `webfs` . . . 38
21.9 Internationalization . . . 38

22 Conclusion **39**
22.1 Putting it all together . . . 39
22.2 Patterns and techniques . . . 39
22.3 Connections to other books . . . 40
22.4 What's missing: CSS, JavaScript, modern web . . . 40
22.5 Comparison with abaco . . . 41
22.6 Modern browser scale . . . 41

A Debugging **43**
A.1 Logging . . . 43
A.2 Per-subsystem tracing . . . 43
A.3 Dumpers . . . 43
A.3.1 URLs . . . 43
A.3.2 HTML items . . . 43
A.3.3 Cookie jar . . . 43

B Profiling **44**

C Error Management **45**
C.1 Per-subsystem error paths . . . 45
C.2 9P error strings across the `webfs` boundary . . . 45

D Utilities **46**
D.1 Strings . . . 46
D.2 Files . . . 46
D.3 Dates . . . 46

E Extra Code **47**
E.1 `webfs/` . . . 47
E.2 `mothra/` . . . 132
E.3 `misc/` . . . 215
E.4 `tcs/` . . . 273

Glossary **309**

Indexes **310**

References **310**

Chapter 1

Introduction

The goal of this book is to explain with full details the source code of a web browser.

1.1 Motivations

Why a web browser? Because I think you are a better programmer if you fully understand how things work under the hood, and the web browser is the program through which most computer users now reach the rest of the software stack. A web browser sits above the UNIX-like user-space API (see the LIBCORE book [Pad16a]), above a windowing system (see the WINDOWS book [Pad16c] and the WIDGETS book [Pad26]), above the network stack (see the NETWORK book [Pad16b]), and above the kernel (see the KERNEL book [Pad14]). It exercises almost every layer of a modern operating system in one program: a parser, an interpreter of a text protocol, a concurrent I/O engine, a rendering engine, and an event-driven GUI.

Most users spend hours every day inside a web browser: reading, writing, watching, shopping, working. Modern engines—Gecko (Firefox), WebKit (Safari), Blink (Chrome)—have become some of the most complex pieces of software ever written, each counting in the millions of lines of code. They implement full CSS box layouts, JavaScript virtual machines with just-in-time compilers, GPU-accelerated compositing, sandboxing, and dozens of other features accreted over three decades. It is no longer realistic for a single programmer to understand all of that in one lifetime.

Yet the essence of a browser is still very simple: fetch a document over a network protocol, parse a markup language, render the result, follow hyperlinks on demand. That essence, stripped of the modern accretions, is what `mothra`—the Plan 9 web browser I will describe in this book—actually implements, in about 4400 lines of C.

There are surprisingly few books explaining how a web browser works. The best I know of is *Web Browser Engineering* [**?**], an online textbook that builds a toy browser in Python. Principia Softwarica complements that by showing the full source code of a real browser—one that you can still launch today on Plan 9 and use to read real web pages, including over HTTPS.

Here are a few questions I hope this book will answer:

- What happens when the user types a URL in the address bar? What is the trace of such a request through the different layers of the software stack?
- What does HTTP look like in practice, at the byte level? How does HTTPS add confidentiality on top?
- How is HTML turned into something you can see and click on? Does mothra build a DOM tree? If not, what does it build instead?
- Why does Plan 9 expose HTTP as a filesystem (`/mnt/web`) rather than as a library to link? What does that design buy over a conventional `libcurl`?

- How are cookies stored and scoped? Why is `webcookies` a separate file server from `webfs`?
- How does a browser handle character encodings (UTF-8, Latin-1, Shift-JIS, GB2312, ...) without getting tangled in charset logic?
- How are images fetched, decoded, and scaled? Why do mothra's images go through three cooperating subprocesses?
- What does mothra not do, and why? No CSS, no JavaScript, no DOM—is such a browser still useful in 2026?

1.2 The Plan 9 web browser: `mothra`

I will explain in this book the code of the Plan 9 web browser `mothra` [?][1], which contains about 4400 lines of C code (LOC), together with the supporting file servers and pipeline tools it relies on: `webfs` (4500 LOC), `webcookies` (1300 LOC), `hget` (1500 LOC), `tcs` (charset conversion), `uhtml` (HTML charset preprocessor), `resize` (image scaling), and `getpix` (image subprocess fetcher).

Like for most books in Principia Softwarica, I chose a Plan 9 program because those programs are simple, small, elegant, open source, and they form together a coherent set. Moreover, `mothra` is arguably simpler to use and to understand than any of the modern browser engines, or even than text-mode browsers like `lynx` [?] or `links` [?] (the most popular text browsers on UNIX).

`mothra` descends from `webpage`, one of the very first graphical web browsers, written by Tom Duff at Bell Labs shortly after Tim Berners-Lee released the first description of the web in 1991. David Hogan rewrote `webpage` as `mothra` in the mid-1990s for Plan 9 4, and `mothra` has been maintained since by the Plan 9 community, most recently by the 9front fork, which keeps it building against current Plan 9 and has added support for TLS 1.2, redirects, and cookies.

Despite its age, `mothra` still works: you can point it at a real HTTPS URL today and it will fetch the page, parse the HTML, and render the text. That is a remarkable property for a 30-year-old program, and one this book will try to explain. The short version is that `mothra` delegates nearly everything that changes (TLS cipher suites, HTTP header trends, cookie standards) to helper programs and file servers beneath it, so the browser code itself does not need to move.

1.3 Other web browsers

Web browsers fall into three broad families: text-mode browsers, classic graphical browsers, and modern engines.

The text-mode family—`lynx` (1992), `links` (2000), `w3m` (1995)—renders HTML as reflowed text on a terminal. They are small (20–80 kLOC), fast, usable over SSH, and good at reading documents. They share with `mothra` the fundamental simplification of treating HTML as a flow of text with inline markup rather than a box-model layout. Where `mothra` differs is that it runs inside a graphical windowing system (see the WINDOWS book [Pad16c]) and can display inline images; the text browsers cannot.

The classic graphical family started with NCSA Mosaic (1993), the first browser to popularize inline images. Netscape Navigator (1994) commercialized Mosaic's design, Microsoft Internet Explorer (1995) cloned Netscape, and the two fought the "browser wars" of the late 1990s. All of these browsers were roughly in `mothra`'s weight class (tens of thousands of LOC) and all implemented HTML 2 or HTML 3.2 without JavaScript or CSS—exactly the subset `mothra` still handles.

The modern family consists of three rendering engines: Gecko (powering Firefox, from Netscape's 1998 open-source release), WebKit (powering Safari, forked from KHTML in 2001), and Blink (powering Chrome, Edge, Opera, forked from WebKit in 2013). Each engine counts in the millions of LOC and implements the full modern

[1] See http://man.9front.org/1/mothra for its manual page.

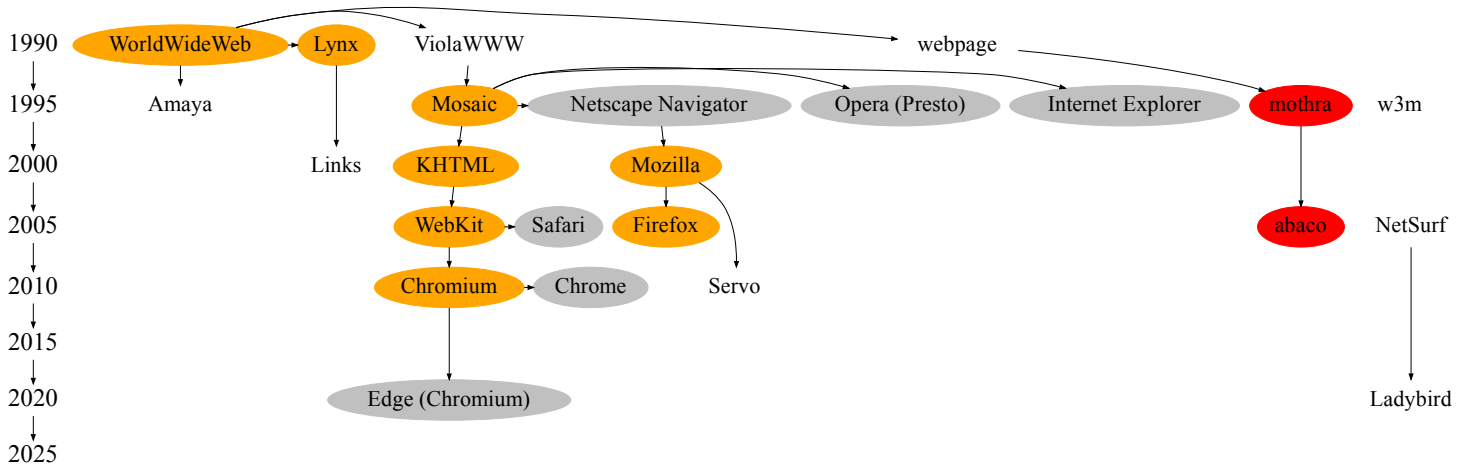

Figure 1.1: Web browsers timeline

web platform: CSS Grid and Flexbox layouts, a JavaScript engine (V8, SpiderMonkey, JavaScriptCore) with JIT compilation, WebGL, WebAssembly, sandboxing, and much more. A good engineer can understand a part of one of these engines in a year of work; nobody understands all of them.

Plan 9 itself has had more than one browser. `mothra`'s predecessor was `webpage` by Tom Duff; its contemporary is `abaco` by Federico Benavento, written in 2004 on top of `libdraw` directly (rather than `libpanel`, which `mothra` uses). `abaco` has slightly better HTML 4 coverage through its use of `libhtml`, a more thorough parser library written by Russ Cox. I will compare the two in the Conclusion.

Figure 1.1 presents a timeline of major web browsers. I think `mothra` represents the best compromise for this book: it implements the essential features of a graphical web browser while still having a small and understandable codebase (about 4400 LOC), and it composes cleanly with the other Plan 9 tools covered elsewhere in Principia Softwarica.

1.4 Getting started

To play with `mothra`, you will first need to install the Plan 9 fork used in Principia Softwarica (see `https://www.principia-softwarica.org/getting-started.html`). Once booted, launch `rio` (the Plan 9 windowing system, see the WINDOWS book [Pad16c]), start a new shell window, and type:

```
% mothra http://example.com
```

A new window will open showing the rendered page. Click on any link to follow it; the address bar updates and mothra fetches the new page. The arrow buttons navigate history; the text box lets you type a fresh URL.

You can also pipe HTML directly into `mothra` for testing:

```
% echo '<h1>Hello</h1><p>World' | mothra
```

This is useful for driving the parser without a network.

1.5 Requirements

This book assumes a good knowledge of the C programming language [**?**]. If that is not the case I suggest you first read *The C Programming Language* [**?**] and possibly also *Advanced Programming in the Unix Environment* [**?**].

Basic familiarity with HTTP and HTML is helpful but not required. The NETWORK book [Pad16b] provides the necessary background on TCP/IP and Plan 9's `/net` filesystem. The KERNEL book [Pad14] explains how

the kernel exposes network devices (including `/net/tls` for TLS) and how namespaces work, which is the substrate on top of which `webfs` operates. The LIBCORE book [Pad16a] covers the user-space C library: `fork`, `exec`, `pipe`, `print`, and the 9P client library that `webfs` uses to serve `/mnt/web`. The WIDGETS book [Pad26] describes `libpanel`, the widget toolkit mothra uses to lay out its address bar, display area, status bar, and form widgets.

This is not an introduction to the web or to HTML. Readers wanting a gentler walk through how a browser works from first principles can read *Web Browser Engineering* [**?**] in parallel with this book.

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

1.8 Acknowledgments

I would like to acknowledge of course Tom Duff, who wrote `webpage`, the ancestor of `mothra`, and David Hogan, who rewrote it as the program I will describe in this book—both wrote in some sense most of this book. I also want to acknowledge Russ Cox, author of `webfs` and `libhtml`; the 9front maintainers who keep `mothra` building against modern Plan 9 and who added TLS, redirect, and cookie support; and Federico Benavento, author of `abaco`, whose work I will contrast with `mothra`'s in the Conclusion.

Chapter 2

Overview

Before showing the source code of `mothra`, `webfs`, and the supporting tools in the following chapters, I first give an overview in this chapter of the general principles of a web browser. I also describe the interfaces that make up the Plan 9 web stack (the `/mnt/web`, `/mnt/webcookies`, and `/net/tls` file servers), walk through a complete `hello.html` document, explain how the code is organized across several directories, and give the background necessary to understand the code I will show later.

2.1 Web browser principles

The following sections explain the core concepts any web browser must address: what a hypertext viewer is, how resources are named, how they are fetched, how they are rendered, how the Plan 9 design composes small programs to do the job, and what `mothra` deliberately leaves out.

2.1.1 A viewer for hypertext documents

A web browser is, at its core, a viewer for a particular kind of document: hypertext. What distinguishes hypertext from ordinary text is the hyperlink: a piece of marked-up text whose activation fetches and displays another document, potentially somewhere else on the network. Stripped of the modern layers (styling, scripting, interactivity), a web browser is then three things glued together: a fetcher (given a name, get the bytes), a parser (given bytes, find the text and the links), and a renderer (given text and links, show them on screen and let the user click). `mothra` implements exactly these three and nothing more.

2.1.2 The URL as universal name

A URL (Uniform Resource Locator) is the universal name of a resource on the web. It has the shape `scheme://host:p` where the scheme (usually `http` or `https`) selects the protocol, the host and port identify the server, the path identifies the resource within the server, the query carries parameters, and the fragment names an anchor inside the document. URLs are the input to nearly every browser operation: the address bar accepts them, hyperlinks embed them, bookmarks record them.

URLs can be absolute (they specify the scheme and host) or relative (they omit those and are resolved against the URL of the current page). Resolving relative URLs is a more subtle operation than it first appears—it involves dot-segment normalization, handling of paths versus authorities, and careful distinction between the URL of the document and the `<base>` declared inside it. I devote Chapter 5 entirely to URL parsing and resolution.

2.1.3 Client/server over a text protocol

A browser fetches resources by speaking HTTP [?], a request/response protocol carried over TCP (or TLS over TCP, for HTTPS). HTTP is remarkable for being a text protocol: a request is literally a few human-readable lines, as is a response. You can see the whole thing by running:

```
% hget -v http://example.com/
```

The server's response is a status line, a list of `Name: value` headers, a blank line, and the body. No binary framing, no TLVs, no schemas. This is one reason the web grew so fast: anybody could debug an HTTP exchange with `telnet` and a keyboard. Plan 9's `hget`, which I describe in Chapter 6, is not much more than `telnet` plus URL parsing plus redirect handling.

2.1.4 Rendering text with structure (no layout engine required)

Once the bytes arrive, the browser has to turn them into something the user can see. Modern browsers use a box model: every HTML element becomes a rectangular box, boxes nest and flow according to CSS rules, and a layout engine (thousands of lines of code) computes positions and sizes. This is how a page with a three-column grid, a sticky header, and a pop-up menu gets laid out.

`mothra` does none of that. It treats an HTML document as a linear flow of text with inline markup, much like a word processor: the tokens come out of the parser and are word-wrapped into paragraphs, with occasional images, form fields, or links embedded in the flow. There is no box model, no column layout, no CSS. Most HTML 2 and HTML 3.2 documents render perfectly well this way; modern pages designed around grids and flex boxes render as reflowed text, which is usually still readable if not pretty.

This is the same simplification the text-mode browsers (`lynx`, `links`, `w3m`) make. The lesson `mothra` teaches is that a big portion of what a browser *does* can be done with a flow-text view and nothing more—the box model is an optimization for visually rich layouts, not a requirement for reading documents.

2.1.5 Composition through pipes and file servers

The most distinctive Plan 9 contribution to the browser design is the use of file servers and pipes where other systems use libraries and frameworks. Instead of linking a library like `libcurl` to speak HTTP, mothra opens files under `/mnt/web`; the kernel routes those opens to the `webfs` file server, which does the actual HTTP work. Instead of linking a charset library like `iconv`, mothra pipes its input through `uhtml` (a 200-line program), which in turn pipes through `tcs` (the Plan 9`iconv` equivalent). Instead of forking a decoder into the browser process, mothra spawns a separate `getpix` helper to fetch and decode images.

The benefit is twofold. First, each piece can be replaced independently: 9front updated `libsec` to support modern TLS cipher suites, and `mothra` gained modern HTTPS with no code change whatsoever. Second, each piece is reusable: `hget` and `mothra` share the same `webfs`; any program that wants to read a URL can `open /mnt/web/clone`; charset conversion is available system-wide through `tcs`. Chapter 7 explores this pattern in depth.

2.1.6 Mothra for the pre-CSS, pre-JS web

`mothra` is a browser for the pre-CSS, pre-JavaScript web. It handles HTML tags (paragraphs, headings, lists, anchors, images, forms, tables), follows links, submits forms, stores cookies, and speaks HTTPS. It does not parse CSS. It does not run JavaScript. It does not lay out pages as a box model. It does not handle cross-origin policies, web sockets, service workers, WebAssembly, or any of the many features modern browsers accreted since about 2005.

Why study it, then? Because the pre-CSS, pre-JS web *is* the substrate the modern web is built on top of. HTTP has not fundamentally changed since 1996. HTML tags still mean what HTML 3.2 said they meant.

TCP, TLS, URLs, MIME, cookies, forms: all of that is in `mothra`, in about 4400 lines of C you can read in a weekend. Understanding `mothra` gives you a correct mental model for the bottom 90% of what any browser does, with none of the distractions of the top 10%.

2.2 The Plan 9 web stack interfaces

Plan 9's design shows up not through a browser library, but through a small set of filesystem interfaces that cooperating programs read and write. The ones relevant to the browser are listed below; each will get a full chapter later.

2.2.1 `/mnt/web` (`webfs`)

`/mnt/web` is the file tree served by `webfs`. It turns HTTP (and HTTPS, and FTP) into filesystem I/O. A program that wants to fetch a URL opens `/mnt/web/clone` (allocating a new connection), writes the URL to the connection's `ctl` file, and reads the resulting body, headers, and metadata from companion files like `body`, `headers`, `contenttype`, `redirect`, and so on. No program on Plan 9 links `libcurl`. They all use `/mnt/web`.

2.2.2 `/mnt/webcookies`

`/mnt/webcookies` is a second file tree served by the `webcookies` program. It persists cookies across `webfs` restarts and across browsers. A subtle point, covered in Chapter 8, is that `webfs` also has its own in-memory cookie store (in `webfs/cookies.c`). Why both? Because `webfs` typically runs per-window and its in-memory cookies die with it; `webcookies` runs once per user session and survives.

2.2.3 Pipelines: `fetch | uhtml | render`

Mothra does not parse charsets, and its HTML parser assumes UTF-8. Many pages on the web are served in other encodings—Shift-JIS, GB2312, ISO-8859-1—and mothra deals with this through a pipeline: the fetched bytes are piped through `uhtml`, a tiny program that sniffs the declared charset and, if needed, shells out to `tcs` to transcode to UTF-8. Only then do the bytes reach mothra's parser. Chapter 10 shows the pipeline in detail.

2.2.4 `/net/tls` and HTTPS

HTTPS (HTTP over TLS) is not implemented in `webfs` either. It is implemented by the Plan 9 kernel as a device driver: `/net/tls` is a file server served by `devtls.c` (see the KERNEL book [Pad14]). To send TLS records, `webfs` opens `/net/tls/clone`, negotiates a session via `libsec`, and then writes plaintext: the kernel does the encryption. This splits the browser's HTTPS story cleanly between a small `libsec` client (thousands of LOC, in user space, replaceable) and a simple I/O loop on plaintext files (unchanged since the 1990s).

2.3 `mothra` command-line interface

The `mothra` command accepts a URL and a handful of flags:

```
usage: mothra [-d] [-m mtpt] [-a] [-v] [url]
```

The `-d` flag enables debug tracing (each fetched URL is logged, and images piped through `resize` are saved to `/tmp/mothra-*.bit` for inspection). The `-m` flag overrides the mount point for `webfs` (default `/mnt/web`), useful for testing. The `-a` flag starts mothra in an "ask for URL" mode (no initial page). The `-v` flag prints the version and exits. If no URL is given, mothra reads HTML from standard input, which is handy for testing the parser.

2.4 hello.html

To make the discussion concrete, consider the simplest meaningful HTML document, `hello.html`:

```
<html>
  <head>
    <title>Hello, Principia</title>
  </head>
  <body>
    <h1>Hello, web!</h1>
    <p>This is a <b>tiny</b> document with a
       <a href="http://example.com/">link</a>
       and an image: <img src="logo.png">.
    </p>
  </body>
</html>
```

Rendered by mothra, this becomes: a heading in bold type, a paragraph of reflowed text with *tiny* shown in bold, `link` shown as an underlined clickable anchor, and `logo.png` fetched by `getpix` and inlined as a bitmap. Follow the `link` and mothra opens a new connection through `/mnt/web`, fetches `http://example.com/`, parses the returned HTML, and paints it in the same window. The back arrow returns here.

Every detail in the preceding paragraph—the flow of text, the discovery of the image URL, the spawning of `getpix`, the `/mnt/web` interaction, the back/forward history—will be explained in a separate chapter. The skeleton of the whole story is in Figure 2.1, below.

2.5 Code organization

Table 2.1 presents short descriptions of the source files that make up the Plan 9 browser stack, together with the main entities (e.g., structures, functions, globals) the file defines, and the corresponding chapter in this document in which the code contained in the file is primarily discussed.

2.6 Software architecture

Figure 2.1 shows the Plan 9 browser stack. Each box is a program or file server; arrows are file I/O or process spawns.

2.6.1 Trace of a web request: `mothra http://example.com`

The traversal of the stack for a single page load is:

1. Mothra's `main()` calls `url2target` to canonicalize `http://example.com` into an internal URL object, using `mothra/url.c` (Chapter 5).

2. Mothra opens `/mnt/web/clone`. The kernel routes the open to `webfs`, which allocates a new `Client` structure and returns a connection number (Chapter 7).

3. Mothra writes the URL to `/mnt/web/$n$/ctl`. Webfs parses it with its full RFC 3986 parser (`webfs/url.c`), opens `/net/tcp/clone` (or `/net/tls/clone` for HTTPS), and sends the HTTP request (Chapter 7).

4. Mothra opens `/mnt/web/$n$/contenttype` and reads the MIME type. It uses `snoop.c` to dispatch (Chapter 12).

Component	Ch.	File	Main entities	LOC
GUI entry point	4	`mothra/mothra.c`	`main()`, event loop, panels	1280
HTML parser	11	`mothra/rdhtml.c`	`rdhtml()`, tag dispatch, `Item`	1278
MIME sniffing	12	`mothra/snoop.c`	`snoop()`, `mimetotype()`	132
Forms	17	`mothra/forms.c`	`Field`, `uencodeform`, `mencodeform`	764
Image fetcher	16	`mothra/getpix.c`	`getpix()`, subprocess pipeline	191
URL adapter	5	`mothra/url.c`	`urlload`, `urlstr`	257
HTML tag table	11	`mothra/html.syntax.c`	tag name table	95
Web file server	7	`webfs/main.c`	`main()`, 9P dispatch	67
9P files	7	`webfs/fs.c`	`/mnt/web` tree	616
HTTP transport	7	`webfs/http.c`	request/response, auth, redirects	539
Per-client thread	7	`webfs/client.c`	`Client`, `clientbodyopen()`	437
Async I/O	7	`webfs/io.c`	`iotlsdial`, `ioprint`	89
Cookie jar (inline)	8	`webfs/cookies.c`	`Jar`, `Cookie`	1173
Plumb dispatch	19	`webfs/plumb.c`	plumb messages	165
URL algebra	5	`webfs/url.c`	RFC 3986	1092
Cookie file server	8	`misc/webcookies.c`	standalone `Jar` server	1266
CLI HTTP client	6	`misc/hget.c`	`doftp`, `dohttp`	1502
HTML charset preprocessor	10	`misc/uhtml.c`	sniff meta, exec `tcs`	194
Image scaling	16	`misc/resize.c`	bilinear resize	224
Charset transcoder	10	`tcs/tcs.c`	conversion driver	596

Table 2.1: Source files of the Plan 9 browser stack covered in this book. Tables of charset-specific data (big5, gbk, jis, ksc, ...) in `tcs/` are not listed.

5. Mothra opens `/mnt/web/$n$/body` and reads the bytes through a pipe to `uhtml`, which reads them back, sniffs the charset, and pipes through `tcs` if needed (Chapter 10).
6. The UTF-8 bytes reach mothra's `rdhtml.c`, which tokenizes the HTML, dispatches on tags, and builds a flat list of `Items` (Chapter 11).
7. The `Item` list is walked by the layout code, which word-wraps it into text runs and emits libpanel draw calls (Chapter 13 and Chapter 14).
8. When an `<img>` tag is encountered, mothra `forks` a `getpix` subprocess and passes it a handle on the new `/mnt/web/$n$/body`. `getpix` decodes the image (possibly piping through `resize` if it is too large) and writes a bitmap that mothra reads back and blits onto the panel (Chapter 16).
9. The user clicks a link. Libpanel calls the anchor handler, which extracts the target URL, resolves it against the base, and goes back to step 2.

2.7 Book structure

The book is organized in three parts, preceded by a short preface of setup chapters and followed by a conclusion and the usual appendices.

The preface (Chapters 1–4) contains this chapter, the core data structures used throughout the code, and a walkthrough of mothra's `main()` and top-level event loop. Reading `main()` early gives you the skeleton on which every later chapter hangs.

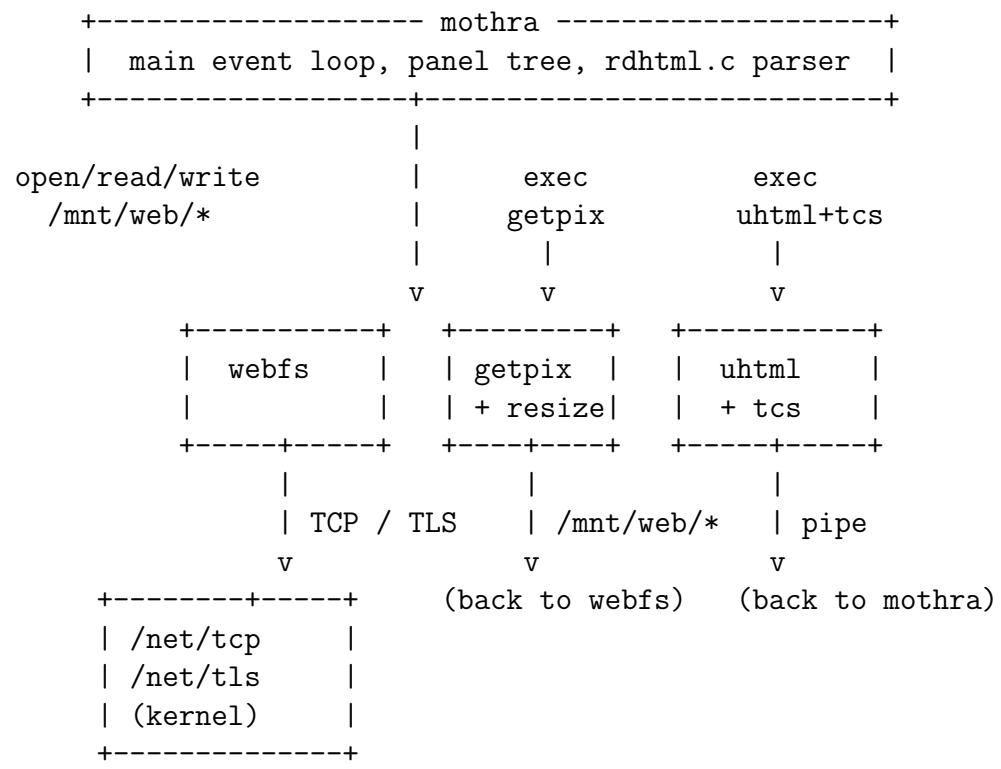

Figure 2.1: The Plan 9 browser stack. `mothra` is the user-facing GUI; `webfs` is a file server exposing HTTP/HTTPS/FTP as `/mnt/web`; `getpix` and `resize` are image helpers; `uhtml` and `tcs` are charset preprocessors; `/net/tcp` and `/net/tls` are kernel devices.

Part I, Retrieving the Page, covers everything that happens before mothra has any bytes to render: URL parsing (Chapter 5), the warmup HTTP client `hget` (Chapter 6), the `webfs` file server (Chapter 7), the `webcookies` companion (Chapter 8), the retrieval state machine inside mothra (Chapter 9), and the charset pipeline (Chapter 10). By the end of Part I, clean UTF-8 bytes are queued up for the parser.

Part II, Rendering the Page, picks up the bytes and turns them into pixels: HTML parsing (Chapter 11), MIME-based content dispatch (Chapter 12), the layout engine (Chapter 13), the per-tag rendering rules (Chapter 14), tables (Chapter 15), images (Chapter 16), and forms (Chapter 17). By the end of Part II, the page is on screen and the user can interact with it.

Part III, Beyond the Basics, covers navigation and history (Chapter 18), event handling and integration with the rest of Plan 9 (Chapter 19), developer tools (Chapter 20), and advanced topics like proxies, gzip, and FTP (Chapter 21).

The conclusion (Chapter 22) summarizes what has been covered, highlights the design patterns that recur across the stack, and compares mothra with `abaco` and with modern browsers. The appendices cover cross-cutting concerns—debugging, profiling, error management, and shared utilities.

Chapter 3

Core Data Structures

3.1 URLs

3.2 HTTP requests and responses

3.3 The `/mnt/web` interface

3.4 Cookies (`Jar`, `Cookie`)

3.5 Charset tables

3.6 HTML parse tree (mothra's flat item list)

3.7 Mothra's view (text runs, layout, panels)

Chapter 4

`mothra`: `main()` and Panels

4.1 Command-line arguments

4.2 Panel layout (address bar, display, status)

4.3 The main I/O loop

4.4 Reading from a URL or stdin

Part I

Retrieving the Page

Chapter 5

Resolving URLs

5.1 Webfs's URL algebra (`webfs/url.c`, RFC 3986)

5.1.1 Parsing absolute URLs (scheme, authority, path, query, fragment)

5.1.2 Authority and userinfo

5.1.3 Resolving relative URLs against a base

5.1.4 Path normalization (dots)

5.1.5 Percent-encoding

5.1.6 Form-data encoding

5.2 Mothra's URL adapter (`mothra/url.c`)

Chapter 6

An HTTP Client: `hget`

6.1 `main()`

6.2 Connecting (TCP and TLS)

6.3 Building the request (`GET`, `HEAD`)

6.4 Reading the response (status, headers, body)

6.5 Following redirects

6.6 Range requests and resume

Chapter 7

The Web File Server: `webfs`

7.1 `main()` and the 9P loop

7.2 The `clone`/`ctl`/`body` protocol

7.3 Connection lifecycle

7.4 HTTP transport (`http.c`)

7.4.1 The request (GET, POST, HEAD)

7.4.2 Status codes

7.4.3 Headers (content-type, content-length, content-encoding)

7.5 Authentication (basic, digest, `www-authenticate`)

7.6 Redirect handling and loop detection

7.7 Per-client thread dispatch (`client.c`)

7.8 Async I/O primitives (`io.c`: `iotlsdial`, `ioprint`)

7.9 HTTPS: `tlsClient` and `/net/tls`

7.10 Why a file server for the network

Chapter 8

The Cookie File Server: `webcookies`

8.1 `main()`

8.2 Storage format

8.3 Domain matching and expiry

8.4 The dual cookie architecture: `webfs/cookies.c` vs standalone `webcookies`

8.5 Why split from `webfs`

Chapter 9

Retrieval and Progress

9.1 Opening `/mnt/web/clone` and writing the URL to `ctl`

9.2 Reading status, headers, and body

9.3 The retrieval state machine

9.4 Progress tracking and status-bar updates

9.5 Scheduling concurrent fetches (main document + images)

9.6 Lifecycle: abort, refresh, redisplay

Chapter 10

Charset Conversion: `tcs` and `uhtml`

10.1 Why charset conversion lives outside the browser

10.2 `tcs` as a pipe-friendly iconv (interface, not internals)

10.3 `uhtml`: sniffing and shelling out

10.4 The full pipeline: `fetch | uhtml | render`

Part II

Rendering the Page

Chapter 11

HTML Parsing: `rdhtml.c`

11.1 What mothra parses (HTML4-ish, lenient)

11.2 Tokenization: tags, attributes, text runs

11.3 The tag dispatch table

11.4 Building a flat list of items (no DOM)

11.5 Entities and character references

11.6 Surviving real-world HTML

Chapter 12

Content Dispatch

12.1 MIME sniffing (`snoop.c`)

12.2 MIME detection via `/bin/file -m` subprocess

12.3 The viewer table (HTML, PLAIN, JPEG, GIF, PNG, ...)

12.4 Fallback logic: `image/` and `text/` prefix rules

12.5 Dispatching to the right reader

12.6 Handling non-HTML content

Chapter 13

Laying Out the Page

13.1 From parsed items to on-screen text runs

13.2 Word wrapping and reflow

13.3 Fonts and text styles

13.4 Hyperlinks and their rendering

13.5 Scrolling and repaint

Chapter 14

Tag-by-Tag Rendering

14.1 Structural (<html>, <head>, <body>)

14.2 Headings and paragraphs

14.3 Text styles (<b>, <i>, <tt>, , ...)

14.4 Lists (<ul>, <ol>, <dl>)

14.5 Breaks and rules (
, <hr>)

14.6 Anchors (<a>)

Chapter 15

Tables

15.1 HTML tables: `<table>`, `<tr>`, `<td>`

15.2 What mothra does: linear pass-through

15.3 Why full table layout is hard (column widths, rowspan, colspan)

15.4 Comparison with abaco and full browsers

Chapter 16

Images

16.1 The `<img>` tag

16.2 `getpix` as a subprocess image fetcher

16.3 The three-stage pipeline: `snooptype()` → format filter → `resize`

16.4 Process management (`rfork`, `waitpid`)

16.5 `resize` for scaling

16.6 Image formats (JPEG/GIF/PNG via `libdraw`)

16.7 Lazy loading and placeholders

16.8 Debug instrumentation (`-d` flag, `poolcheck`, heap dumps)

Chapter 17

Forms

17.1 The `<form>` tag and `forms.c`

17.2 The `Field` state machine (TYPEIN, CHECK, PASSWD, RADIO, ...)

17.3 Input widgets (`<input type=...>`)

17.4 Text areas and select menus

17.5 Password fields

17.6 Form encoding

17.6.1 `uencodeform` (application/x-www-form-urlencoded)

17.6.2 `mencodeform` (multipart/form-data)

17.7 Submission: GET vs POST

Part III

Beyond the Basics

Chapter 18

Navigation and History

18.1 Following links

18.2 Back, forward, and history

18.3 Bookmarks and hotlist

Chapter 19

Events and Integration

19.1 Mouse clicks and scrolling

19.2 Keyboard shortcuts

19.3 The `plumb(2)` protocol

19.4 Plumb messages: MIME type + filename

19.5 Spawning external viewers

19.6 Snarf (selection and clipboard)

Chapter 20

Developer Tools

20.1 View source

20.2 HTTP tracing (`-d` flag)

20.3 MIME-type dumping (`snoop.c`)

20.4 The debug-instrumentation pattern (fork + `poolcheck` + temp files)

20.5 Saving a page

Chapter 21

Advanced Topics

21.1 Proxy (via `$http_proxy`)

21.2 Referer header

21.3 Content encoding (gzip)

21.4 Animated GIFs

21.5 File protocol (`file://`)

21.6 FTP protocol support (`doftp` in hget, ftp in `webfs/url.c`)

21.7 External viewers via plumbing

21.8 Caching in `webfs`

21.9 Internationalization

Chapter 22

Conclusion

You now know how the Plan 9 web browser `mothra` works, to the smallest details, and more generally how many web browsers work at their core. You have followed a URL like `http://example.com/` from the address bar, through mothra's URL parser, through a 9P conversation with `webfs`, through a TCP (or TLS) dial to the origin server, through the charset pipeline `uhtml | tcs`, into mothra's `rdhtml.c` parser, out as a flat list of `Item`s, and finally onto the screen as word-wrapped text with clickable anchors and inlined images. Click a link and the same machinery runs again, one page at a time.

22.1 Putting it all together

At its core, `mothra` solves the browser problem by *refusing to solve most of it*. It does not have a layout engine; it treats documents as a flow of text. It does not have a TLS library; it opens `/net/tls` and writes plaintext. It does not have a cookie subsystem of its own; it delegates to `webfs` and `webcookies`. It does not have an image decoder; it spawns `getpix`. It does not have a charset library; it pipes through `uhtml` and `tcs`.

What is left in `mothra`, once all that is subtracted, is about 4400 lines of C that do the three things a browser cannot avoid doing itself: parse HTML, lay out a flow of text, and drive a user-facing event loop. Everything else is somebody else's job.

22.2 Patterns and techniques

Several techniques appear repeatedly across the stack, and apply far beyond web browsers:

- *File server as API boundary*: `/mnt/web`, `/mnt/webcookies`, and `/net/tls` are all file servers. A program interacts with them through `open`, `read`, and `write`—the same operations it uses for ordinary files. No library needs to be linked, no versioning of APIs is needed: the filesystem is the ABI. This is the Plan 9 design signature and appears throughout the KERNEL book [Pad14] and the LIBCORE book [Pad16a] as well.

- *Pipe composition for transformation*: the `fetch | uhtml | tcs` pipeline is a chain of tiny programs, each doing one thing. It is literally a Plan 9 shell pipeline, dressed up as a browser input stage. The same pattern appears wherever a sequence of transformations is more easily expressed as cooperating processes than as a monolithic routine; the SHELL book [Pad18] develops it in full.

- *Subprocess per helper*: `getpix` is not a library call, it is a separate program mothra `fork`s. This keeps the browser's address space clean of decoder bugs, makes helpers replaceable, and gives each helper its own debug flags. Modern browsers reinvent this pattern in the large: Chrome's per-tab process, its GPU process, and its network process are all helpers-in-processes, with millions of lines of IPC plumbing around them.

- *Content dispatch from MIME type*: `snoop.c` runs `/bin/file -m` as a subprocess, classifies the content, and chooses a renderer. Polymorphism by sniffing, rather than by type declaration. The same idea is behind UNIX's shebang lines, Plan 9's `/mnt/plumb` dispatch, and HTTP's own `Content-Type` negotiation.
- *Lenient parsing*: `rdhtml.c` never refuses a malformed document. This is the same approach every real HTML parser takes, because rejecting real-world HTML would reject most of the web. The lesson—that a public data format, once popular, cannot be made strict after the fact—recurs in HTTP header parsing, email, URLs, CSV, JSON in the wild, and every other popular text format.
- *Flat item list instead of a DOM*: mothra does not build a tree you can query after the fact, because it never needs to. Removing the DOM removes tens of thousands of lines of bookkeeping; that is the price the modern web pays for letting JavaScript inspect the document. The general lesson: if nobody *queries* your parsed representation later, you do not need an inspectable tree.

22.3 Connections to other books

Because mothra is a user-space graphical program that talks to a file-server network stack, it sits at the top of several books in Principia Softwarica, and I encourage you to continue with any of them:

- The KERNEL book [Pad14] explains what lies behind the file servers mothra uses: `/net/tcp`, `/net/tls` (`devtls.c`), the namespace mechanism that attaches `/mnt/web` into the process's view of the filesystem, and the `rfork` syscall mothra uses to spawn `getpix`.
- The LIBCORE book [Pad16a] explains the user-space C library underneath: `print`, `Bio` buffered I/O, `exec`, `pipe`, the 9P client library `lib9p` that both mothra and `webfs` use, and the memory allocator whose debug hooks are exposed by `mothra -d`.
- The WIDGETS book [Pad26] explains `libpanel`, the toolkit mothra uses for its address bar, display pane, status bar, and form widgets. Form fields in particular are ordinary libpanel widgets (text entry, check box, radio group) assembled by `forms.c`.
- The WINDOWS book [Pad16c] explains `rio`, the Plan 9 window manager mothra runs under. Mothra's event loop, plumb integration, and snarf (selection) behavior only make sense in the context of rio's conventions.
- The NETWORK book [Pad16b] explains the TCP/IP stack, the `/net` filesystem, DNS resolution, and TLS—everything `webfs` ultimately delegates to the kernel for.
- The SHELL book [Pad18] is, somewhat unexpectedly, relevant to the browser: `uhtml` literally execs `rc` to run a `\tcs -f cset || cat\ | tcs -f html` pipeline. The charset stage of the browser is a shell pipeline.

22.4 What's missing: CSS, JavaScript, modern web

What you will not find in `mothra`:

- CSS. No selectors, no cascade, no box model. Styling is implicit in mothra's per-tag rendering defaults (headings get bold larger fonts, `<code>` gets a fixed-width font, and so on). Sites that rely on CSS for layout will render as linearized text.
- JavaScript. No script execution, no DOM, no event model beyond the browser-native clicks and keyboard. Pages that generate their content in JavaScript after load (single-page apps) will appear mostly empty.

- Cross-origin policies, CORS, service workers, web sockets, WebAssembly, WebGL, indexed DB, fetch API, and the long tail of modern web platform features. None of this exists.
- Modern security model: mothra will happily submit a form to any URL, cross-origin or not. There is no concept of a secure context beyond HTTPS itself.
- Incremental layout: mothra fetches the full document, parses it, and renders. It does not progressively update the page as bytes arrive. (It does progressively fetch images in parallel, however.)

Whether any of this matters for you depends on what you want the browser for. For reading documents on the web, mothra still works surprisingly well in 2026, especially on text-heavy sites. For using web applications—which is what most users today mean by "the web"—it does not.

22.5 Comparison with abaco

Plan 9 has another graphical browser, `abaco`, written in 2004 by Federico Benavento. The two make different design choices and are worth comparing.

- Size: `mothra` is about 4400 LOC; `abaco` is about 7900 LOC. `abaco` also depends on `libhtml`, a separate 4500 LOC HTML library by Russ Cox, which `mothra` does not use.
- Parser: `abaco` uses `libhtml`, a more thorough HTML 4 parser with better support for tables, styles, and HTML entities. `mothra` has its own parser in `rdhtml.c`, 1300 LOC, that trades coverage for simplicity.
- UI toolkit: `mothra` uses `libpanel` (see the WIDGETS book [Pad26]), which gives it integration with the rest of the Plan 9 widget ecosystem. `abaco` has its own UI layer written directly on `libdraw`, with support for tabs, incremental layout, and a more modern event model.
- Composition: `mothra` uses the `uhtml | tcs` pipeline for charset handling, spawned as a subprocess. `abaco` links `tcs` in directly, with its own `charsets.txt`/`charsets.awk` generator. Mothra's is the more Plan 9-native design; abaco's is more self-contained.
- Maintenance: `mothra` is still carried and patched by the 9front fork of Plan 9. `abaco` has been frozen since its original release, with only cosmetic changes since.

I chose `mothra` for this book because it is smaller, because its composition through pipes and file servers makes it a cleaner case study of Plan 9 design principles, and because `libpanel` integration gives us a natural cross-reference to the WIDGETS book [Pad26]. `abaco` is more architecturally ambitious but tells a muddier design story.

22.6 Modern browser scale

For perspective, here are rough sizes of modern web engines compared to `mothra`:

The three orders of magnitude between `mothra` and Chromium are not gratuitous: they buy CSS layout, a JavaScript engine with JIT, GPU compositing, a sandbox, WebAssembly, Web APIs, modern cryptography, and thousands of quirks accreted over 30 years of web compatibility. But they also hide the essential structure that is plainly visible in `mothra`: URL, HTTP, HTML, flow layout, click to follow.

The core algorithm of a browser has not changed since 1991. What changed is what a browser is expected to do *besides* displaying hypertext. Reading `mothra` is a way to see that core without the accretions—to understand, from end to end, what is actually necessary to turn a URL into a clickable page. Everything else, as far as pedagogy is concerned, is an optimization.

Engine / browser	**LOC (approx.)**
mothra + webfs + libhtml + helpers (this book)	14 k
Links	85 k
Lynx	170 k
Gecko (Firefox's engine)	6 M
WebKit (Safari's engine)	9 M
Chromium (Chrome, Edge, ...)	32 M

Table 22.1: Approximate lines of code of web browsers (2026).

Appendix A

Debugging

A.1 Logging

A.2 Per-subsystem tracing

A.3 Dumpers

A.3.1 URLs

A.3.2 HTML items

A.3.3 Cookie jar

Appendix B

Profiling

Appendix C

Error Management

C.1 Per-subsystem error paths

C.2 9P error strings across the `webfs` boundary

Appendix D

Utilities

D.1 Strings

D.2 Files

D.3 Dates

Appendix E

Extra Code

E.1 webfs/

webfs/dat.h

⟨*struct* Ibuf *(webfs)* 47a⟩≡ (48c)

```
/* simple buffered i/o for network connections; shared by http, ftp */
struct Ibuf
{
    int fd;
    Ioproc *io;
    char buf[4096];
    char *rp, *wp;
};
```

⟨*struct* Ctl *(webfs)* 47b⟩≡ (48c)

```
struct Ctl
{
    int acceptcookies;
    int sendcookies;
    int redirectlimit;
    char        *useragent;
};
```

⟨*struct* Client *(webfs)* 47c⟩≡ (48c)

```
struct Client
{
    Url *url;
    Url *baseurl;
    Ctl ctl;
    Channel *creq;      /* chan(Req*) */
    int num;
    int plumbed;
    char *contenttype;
    char *postbody;
    char *redirect;
    char *authenticate;
    char *ext;
    int npostbody;
    int havepostbody;
    int iobusy;
    int bodyopened;
    Ioproc *io;
    int ref;
    void *aux;
};
```

⟨*enum* `UScheme` 48a⟩≡ (48c)

```
/*
 * If ischeme is USunknown, then the given URL is a relative
 * URL which references the "current document" in the context of the base.
 * If this is the case, only the "fragment" and "url" members will have
 * meaning, and the given URL structure may not be used as a base URL itself.
 */
enum
{
    USunknown,
    UShttp,
    UShttps,
    USftp,
    USfile,
    UScurrent,
};
```

⟨*struct* `Url` *(webfs)* 48b⟩≡ (48c)

```
struct Url
{
    int         ischeme;
    char*       url;
    char*       scheme;
    int         (*open)(Client*, Url*);
    int         (*read)(Client*, Req*);
    void                (*close)(Client*);
    char*       schemedata;
    char*       authority;
    char*       user;
    char*       passwd;
    char*       host;
    char*       port;
    char*       path;
    char*       query;
    char*       fragment;
    union {
        struct {
            char *page_spec;
        } http;
        struct {
            char *path_spec;
            char *type;
        } ftp;
    };
};
```

⟨`webfs/dat.h` 48c⟩≡

```
typedef struct Client Client;
typedef struct Ctl Ctl;
typedef struct Ibuf Ibuf;
typedef struct Url Url;

⟨struct Ibuf (webfs) 47a⟩

⟨struct Ctl (webfs) 47b⟩

⟨struct Client (webfs) 47c⟩

⟨enum UScheme 48a⟩
```

⟨*struct* `Url` *(webfs)* 48b⟩

```
enum
{
    STACK = 32*1024,  /* was 16*1024; there are big arrays on the stack */
};

extern  Client**        client;
extern  int             cookiedebug;
extern  Srv             fs;
extern  int             fsdebug;
extern  Ctl             globalctl;
extern  int             nclient;
extern  int             urldebug;
extern  int             httpdebug;
extern  char*   status[];
```

webfs/fns.h

⟨`webfs/fns.h` 49⟩≡

```
/* buf.c */
void            initibuf(Ibuf*, Ioproc*, int);
int             readibuf(Ibuf*, char*, int);
void            unreadline(Ibuf*, char*);
int             readline(Ibuf*, char*, int);

/* client.c */
int             newclient(int);
void            closeclient(Client*);
void            clonectl(Ctl*);
int             ctlwrite(Req*, Ctl*, char*, char*);
int             clientctlwrite(Req*, Client*, char*, char*);
int             globalctlwrite(Req*, char*, char*);
void            ctlread(Req*, Client*);
void            globalctlread(Req*);
void            plumburl(char*, char*);

/* cookies.c */
void            cookieread(Req*);
void            cookiewrite(Req*);
void            cookieopen(Req*);
void            cookieclunk(Fid*);
void            initcookies(char*);
void            closecookies(void);
void            httpsetcookie(char*, char*, char*);
char*   httpcookies(char*, char*, int);

/* fs.c */
void            initfs(void);

/* http.c */
int             httpopen(Client*, Url*);
int             httpread(Client*, Req*);
void            httpclose(Client*);

/* io.c */
int             iotlsdial(Ioproc*, char*, char*, char*, int*, int, char*);
int             ioprint(Ioproc*, int, char*, ...);
```

```
#pragma varargck argpos ioprint 3

/* plumb.c */
void    plumbinit(void);
void    plumbstart(void);
void    replumb(Client*);

/* url.c */
Url*            parseurl(char*, Url*);
void            freeurl(Url*);
void            rewriteurl(Url*);
int             seturlquery(Url*, char*);
Url*            copyurl(Url*);
char*   escapeurl(char*, int(*)(int));
char*   unescapeurl(char*);
void            initurl(void);

/* util.c */
char*   estrdup(char*);
char*   estrmanydup(char*, ...);
char*   estredup(char*, char*);
void*   emalloc(uint);
void*   erealloc(void*, uint);
char*   strlower(char*);
```

webfs/buf.c

⟨*function* `initibuf` *(webfs)* 50a⟩≡ (51b)

```
void
initibuf(Ibuf *b, Ioproc *io, int fd)
{
    b->fd = fd;
    b->io = io;
    b->rp = b->wp = b->buf;
}
```

⟨*function* `readibuf` *(webfs)* 50b⟩≡ (51b)

```
int
readibuf(Ibuf *b, char *buf, int len)
{
    int n;

    n = b->wp - b->rp;
    if(n > 0){
        if(n > len)
            n = len;
        memmove(buf, b->rp, n);
        b->rp += n;
        return n;
    }
    return ioreadn(b->io, b->fd, buf, len);
}
```

⟨*function* `unreadline` *(webfs)* 50c⟩≡ (51b)

```
void
unreadline(Ibuf *b, char *line)
{
    int i, n;
```

```
    i = strlen(line);
    n = b->wp - b->rp;
    memmove(&b->buf[i+1], b->rp, n);
    memmove(b->buf, line, i);
    b->buf[i] = '\n';
    b->rp = b->buf;
    b->wp = b->rp+i+1+n;
}
```

⟨*function* `readline` *(webfs)* 51a⟩≡ (51b)

```
int
readline(Ibuf *b, char *buf, int len)
{
    int n;
    char *p;

    len--;

    for(p = buf;;){
        if(b->rp >= b->wp){
            /* claude: reset on drain -- same upstream ring-buffer bug as
             * hget's readline. wp only advances; without resetting to
             * b->buf after the ring drains, cumulative reads push wp
             * past b->buf+sizeof(b->buf) and the next read() writes off
             * the end of the allocation. Fat-header sites
             * (en.wikipedia.org etc.) trip it. unreadline() already
             * re-anchors rp/wp at b->buf when it fires, so the two
             * mutators stay consistent. */
            b->rp = b->wp = b->buf;
            n = ioread(b->io, b->fd, b->wp, sizeof(b->buf)/2);
            if(n < 0)
                return -1;
            if(n == 0)
                break;
            b->wp += n;
        }
        n = *b->rp++;
        if(len > 0){
            *p++ = n;
            len--;
        }
        if(n == '\n')
            break;
    }

    /* drop trailing white */
    for(;;){
        if(p <= buf)
            break;
        n = *(p-1);
        if(n != ' ' && n != '\t' && n != '\r' && n != '\n')
            break;
        p--;
    }

    *p = 0;
    return p-buf;
}
```

⟨`webfs/buf.c` 51b⟩≡

```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ip.h>
#include <plumb.h>
#include <thread.h>
#include <fcall.h>
#include <9p.h>
#include "dat.h"
#include "fns.h"
```

⟨*function* `initibuf` *(webfs)* 50a⟩

⟨*function* `readibuf` *(webfs)* 50b⟩

⟨*function* `unreadline` *(webfs)* 50c⟩

⟨*function* `readline` *(webfs)* 51a⟩

webfs/client.c

⟨*enum* `Xtype` 52a⟩≡ (59c)
```
enum
{
    Bool,
    Int,
    String,
    XUrl,
    Fn,
};
```

⟨*struct* `Ctab` *(webfs)* 52b⟩≡ (59c)
```
struct Ctab {
    char *name;
    int type;
    void *offset;
};
```

⟨*global* `ctltab` *(webfs)* 52c⟩≡ (59c)
```
Ctab ctltab[] = {
    "acceptcookies",    Bool,          (void*)offsetof(Ctl, acceptcookies),
    "sendcookies",              Bool,          (void*)offsetof(Ctl, sendcookies),
    "redirectlimit",            Int,           (void*)offsetof(Ctl, redirectlimit),
    "useragent",                String, (void*)offsetof(Ctl, useragent),
};
```

⟨*global* `globaltab` *(webfs)* 52d⟩≡ (59c)
```
Ctab globaltab[] = {
    "chatty9p",         Int,           &chatty9p,
    "fsdebug",          Int,           &fsdebug,
    "cookiedebug",              Int,           &cookiedebug,
    "urldebug",         Int,           &urldebug,
    "httpdebug",                Int,           &httpdebug,
};
```

⟨*global* `clienttab` *(webfs)* 52e⟩≡ (59c)
```
Ctab clienttab[] = {
    "baseurl",                  XUrl,          (void*)offsetof(Client, baseurl),
    "url",                              XUrl,          (void*)offsetof(Client, url),
};
```

⟨*function* `newclient` *(webfs)* 53a⟩≡ (59c)

```
int
newclient(int plumbed)
{
    int i;
    Client *c;

    for(i=0; i<nclient; i++)
        if(client[i]->ref==0)
            return i;

    c = emalloc(sizeof(Client));
    c->plumbed = plumbed;
    c->creq = chancreate(sizeof(Req*), 8);
    threadcreate(clientthread, c, STACK);

    c->io = ioproc();
    c->num = nclient;
    c->ctl = globalctl;
    clonectl(&c->ctl);
    if(nclient%16 == 0)
        client = erealloc(client, (nclient+16)*sizeof(client[0]));
    client[nclient++] = c;
    return nclient-1;
}
```

⟨*function* `closeclient` *(webfs)* 53b⟩≡ (59c)

```
void
closeclient(Client *c)
{
    if(--c->ref == 0){
        if(c->bodyopened){
            if(c->url && c->url->close)
                (*c->url->close)(c);
            c->bodyopened = 0;
        }
        free(c->contenttype);
        c->contenttype = nil;
        free(c->postbody);
        c->postbody = nil;
        freeurl(c->url);
        c->url = nil;
        free(c->redirect);
        c->redirect = nil;
        free(c->authenticate);
        c->authenticate = nil;
        c->npostbody = 0;
        c->havepostbody = 0;
        c->bodyopened = 0;
    }
}
```

⟨*function* `clonectl` *(webfs)* 53c⟩≡ (59c)

```
void
clonectl(Ctl *c)
{
    if(c->useragent)
        c->useragent = estrdup(c->useragent);
}
```

⟨*function* clientbodyopen*(webfs)* 54⟩≡ (59c)

```
void
clientbodyopen(Client *c, Req *r)
{
    char e[ERRMAX], *next;
    int i, nauth;
    Url *u;

    nauth = 0;
    next = nil;
    for(i=0; i<=c->ctl.redirectlimit; i++){
        if(c->url == nil){
            werrstr("nil url");
            goto Error;
        }
        if(c->url->open == nil){
            werrstr("unsupported url type");
            goto Error;
        }
        if(fsdebug)
            fprint(2, "try %s\n", c->url->url);
        if(c->url->open(c, c->url) < 0){
        Error:
            if(next)
                fprint(2, "next %s (but for error)\n", next);
            free(next);
            rerrstr(e, sizeof e);
            c->iobusy = 0;
            if(r != nil)
                r->fid->omode = -1;
            closeclient(c);     /* not opening */
            if(r != nil)
                respond(r, e);
            return;
        }
        if (c->authenticate && nauth++ < 1)
                continue;
        if(!c->redirect)
            break;
        next = c->redirect;
        c->redirect = nil;
        if(i==c->ctl.redirectlimit){
            werrstr("redirect limit reached");
            goto Error;
        }
        if((u = parseurl(next, c->url)) == nil)
            goto Error;
        if(urldebug)
            fprint(2, "parseurl %s got scheme %d\n", next, u->ischeme);
        if(u->ischeme == USunknown){
            werrstr("redirect with unknown URL scheme");
            goto Error;
        }
        if(u->ischeme == UScurrent){
            werrstr("redirect to URL relative to current document");
            goto Error;
        }
        freeurl(c->url);
        c->url = u;
    }
```

```
    free(next);
    c->iobusy = 0;
    if(r != nil)
        respond(r, nil);
}
```

⟨*function* `plumburl` *(webfs)* 55a⟩≡ (59c)

```
void
plumburl(char *url, char *base)
{
    int i;
    Client *c;
    Url *ubase, *uurl;

    ubase = nil;
    if(base){
        ubase = parseurl(base, nil);
        if(ubase == nil)
            return;
    }
    uurl = parseurl(url, ubase);
    if(uurl == nil){
        freeurl(ubase);
        return;
    }
    i = newclient(1);
    c = client[i];
    c->ref++;
    c->baseurl = ubase;
    c->url = uurl;
    sendp(c->creq, nil);
}
```

⟨*function* `clientbodyread` *(webfs)* 55b⟩≡ (59c)

```
void
clientbodyread(Client *c, Req *r)
{
    char e[ERRMAX];

    if(c->url->read == nil){
        respond(r, "unsupported url type");
        return;
    }
    if(c->url->read(c, r) < 0){
        rerrstr(e, sizeof e);
        c->iobusy = 0;
        respond(r, e);
        return;
    }
    c->iobusy = 0;
    respond(r, nil);
}
```

⟨*function* `clientthread` *(webfs)* 55c⟩≡ (59c)

```
static void
clientthread(void *a)
{
    Client *c;
    Req *r;
```

```
    c = a;
    if(c->plumbed) {
        recvp(c->creq);
        if(c->url == nil){
            fprint(2, "bad url got plumbed\n");
            return;
        }
        clientbodyopen(c, nil);
        replumb(c);
    }
    while((r = recvp(c->creq)) != nil){
        if(fsdebug)
            fprint(2, "clientthread %F\n", &r->ifcall);
        switch(r->ifcall.type){
        case Topen:
            if(c->plumbed) {
                c->plumbed = 0;
                c->ref--;                       /* from plumburl() */
                respond(r, nil);
            }
            else
                clientbodyopen(c, r);
            break;
        case Tread:
            clientbodyread(c, r);
            break;
        case Tflush:
            respond(r, nil);
        }
        if(fsdebug)
            fprint(2, "clientthread finished req\n");
    }
}
```

⟨*function* `findcmd` *(webfs)* 56a⟩≡ (59c)

```
static Ctab*
findcmd(char *cmd, Ctab *tab, int ntab)
{
    int i;

    for(i=0; i<ntab; i++)
        if(strcmp(tab[i].name, cmd) == 0)
            return &tab[i];
    return nil;
}
```

⟨*function* `parseas` *(webfs)* 56b⟩≡ (59c)

```
static void
parseas(Req *r, char *arg, int type, void *a)
{
    Url *u;
    char e[ERRMAX];

    switch(type){
    case Bool:
        if(strcmp(arg, "on")==0 || strcmp(arg, "1")==0)
            *(int*)a = 1;
        else
            *(int*)a = 0;
        break;
```

```
    case String:
        free(*(char**)a);
        *(char**)a = estrdup(arg);
        break;
    case XUrl:
        u = parseurl(arg, nil);
        if(u == nil){
            snprint(e, sizeof e, "parseurl: %r");
            respond(r, e);
            return;
        }
        freeurl(*(Url**)a);
        *(Url**)a = u;
        break;
    case Int:
        if(strcmp(arg, "on")==0)
            *(int*)a = 1;
        else
            *(int*)a = atoi(arg);
        break;
    }
    respond(r, nil);
}
```

⟨*function* `ctlwrite` *(webfs)* 57a⟩≡ (59c)

```
int
ctlwrite(Req *r, Ctl *ctl, char *cmd, char *arg)
{
    void *a;
    Ctab *t;

    if((t = findcmd(cmd, ctltab, nelem(ctltab))) == nil)
        return 0;
    a = (void*)((uintptr)ctl+(uintptr)t->offset);
    parseas(r, arg, t->type, a);
    return 1;
}
```

⟨*function* `clientctlwrite` *(webfs)* 57b⟩≡ (59c)

```
int
clientctlwrite(Req *r, Client *c, char *cmd, char *arg)
{
    void *a;
    Ctab *t;
    Url *u;
    char e[ERRMAX];

    if((t = findcmd(cmd, clienttab, nelem(clienttab))) == nil)
        return 0;
    /*
     * claude: webfs's ctl interface exposes two commands per clone,
     * `baseurl` and `url`, with both slots stored in the Client struct
     * on the clone's state. The design clearly intended the RFC2396
     * delta-URI flow -- a caller that doesn't itself know how to
     * combine a base with a relative reference writes
     *
     *     baseurl https://en.wikipedia.org/wiki/Main_Page
     *     url     /wiki/Cat
     *
     * and webfs is supposed to resolve the relative form against the
```

```
         * previously-set base. But in principia's inherited code all
         * commands route through the same generic parseas() below, which
         * always calls parseurl(arg, nil). So the `url` write has no
         * access to c->baseurl and the relative form dies with "relative
         * URI given without base". Observable consequence: `baseurl` is
         * written, accepted, stored -- and then silently ignored by the
         * very next `url` write. Only clients that resolve absolute URLs
         * themselves (abaco, hget) work; clients that rely on webfs to
         * combine (mothra) fail on links like <a href="/..."> which is
         * most of the Web.
         *
         * 9front fixed this long ago by rewriting clientctl as a manual
         * dispatch that knows which slot it's writing (see
         * 9front/sys/src/cmd/webfs/fs.c:clientctl). We achieve the same
         * effect with a minimal surgery: special-case `url` here so it
         * passes c->baseurl to parseurl (nil baseurl falls back to the
         * previous behavior, i.e. absolute works, relative fails cleanly).
         */
        if(strcmp(cmd, "url") == 0){
            u = parseurl(arg, c->baseurl);
            if(u == nil){
                snprint(e, sizeof e, "parseurl: %r");
                respond(r, e);
                return 1;
            }
            freeurl(c->url);
            c->url = u;
            respond(r, nil);
            return 1;
        }
        a = (void*)((uintptr)c+(uintptr)t->offset);
        parseas(r, arg, t->type, a);
        return 1;
    }
```

⟨*function* `globalctlwrite` *(webfs)* 58a⟩≡ (59c)

```
    int
    globalctlwrite(Req *r, char *cmd, char *arg)
    {
        void *a;
        Ctab *t;

        if((t = findcmd(cmd, globaltab, nelem(globaltab))) == nil)
            return 0;
        a = t->offset;
        parseas(r, arg, t->type, a);
        return 1;
    }
```

⟨*function* `ctlfmt` *(webfs)* 58b⟩≡ (59c)

```
    static void
    ctlfmt(Ctl *c, char *s)
    {
        int i;
        void *a;
        char *t;

        for(i=0; i<nelem(ctltab); i++){
            a = (void*)((uintptr)c+(uintptr)ctltab[i].offset);
            switch(ctltab[i].type){
```

```
        case Bool:
            s += sprint(s, "%s %s\n", ctltab[i].name, *(int*)a ? "on" : "off");
            break;
        case Int:
            s += sprint(s, "%s %d\n", ctltab[i].name, *(int*)a);
            break;
        case String:
            t = *(char**)a;
            if(t != nil)
                s += sprint(s, "%s %.*s%s\n", ctltab[i].name, utfnlen(t, 100), t, strlen(t)>100 ? "..." : "");
            break;
        }
    }
}
```

⟨*function* ctlread*(webfs)* 59a⟩≡ (59c)

```
void
ctlread(Req *r, Client *c)
{
    char buf[1024];

    sprint(buf, "%11d \n", c->num);
    ctlfmt(&c->ctl, buf+strlen(buf));
    readstr(r, buf);
    respond(r, nil);
}
```

⟨*function* globalctlread*(webfs)* 59b⟩≡ (59c)

```
void
globalctlread(Req *r)
{
    char buf[1024], *s;
    int i;

    s = buf;
    for(i=0; i<nelem(globaltab); i++)
        s += sprint(s, "%s %d\n", globaltab[i].name, *(int*)globaltab[i].offset);
    ctlfmt(&globalctl, s);
    readstr(r, buf);
    respond(r, nil);
}
```

⟨webfs/client.c 59c⟩≡

```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ip.h>
#include <plumb.h>
#include <thread.h>
#include <fcall.h>
#include <9p.h>
#include "dat.h"
#include "fns.h"

int nclient;
Client **client;

static void clientthread(void*);
⟨function newclient(webfs) 53a⟩
```

⟨*function* `closeclient`*(webfs)* 53b⟩

⟨*function* `clonectl`*(webfs)* 53c⟩

⟨*function* `clientbodyopen`*(webfs)* 54⟩

⟨*function* `plumburl`*(webfs)* 55a⟩

⟨*function* `clientbodyread`*(webfs)* 55b⟩

⟨*function* `clientthread`*(webfs)* 55c⟩

⟨*enum* `Xtype` 52a⟩

```
typedef struct Ctab Ctab;
```
⟨*struct* `Ctab`*(webfs)* 52b⟩

⟨*global* `ctltab`*(webfs)* 52c⟩

⟨*global* `globaltab`*(webfs)* 52d⟩

⟨*global* `clienttab`*(webfs)* 52e⟩

⟨*function* `findcmd`*(webfs)* 56a⟩

⟨*function* `parseas`*(webfs)* 56b⟩

⟨*function* `ctlwrite`*(webfs)* 57a⟩

⟨*function* `clientctlwrite`*(webfs)* 57b⟩

⟨*function* `globalctlwrite`*(webfs)* 58a⟩

⟨*function* `ctlfmt`*(webfs)* 58b⟩

⟨*function* `ctlread`*(webfs)* 59a⟩

⟨*function* `globalctlread`*(webfs)* 59b⟩

webfs/cookies.c

⟨*struct* `Cookie`*(webfs)* 60⟩≡ (80c)

```
struct Cookie
{
    /* external info */
    char*       name;
    char*       value;
    char*       dom;            /* starts with . */
    char*       path;
    char*       version;
    char*       comment;                /* optional, may be nil */

    uint                expire;         /* time of expiration: ~0 means when webcookies dies */
    int         secure;
    int         explicitdom;    /* dom was explicitly set */
    int         explicitpath;   /* path was explicitly set */
    int         netscapestyle;
```

```
    /* internal info */
    int         deleted;
    int         mark;
    int         ondisk;
};
```

⟨*struct* `Jar` *(webfs)* 61a⟩≡ (80c)

```
struct Jar
{
    Cookie      *c;
    int         nc;
    int         mc;

    Qid         qid;
    int         dirty;
    char                *file;
    char                *lockfile;
};
```

⟨*global* `stab` *(webfs)* 61b⟩≡ (80c)

```
struct {
    char *s;
    int offset;
    int ishttp;
} stab[] = {
    "domain",           offsetof(Cookie, dom),          1,
    "path",             offsetof(Cookie, path),         1,
    "name",             offsetof(Cookie, name),         0,
    "value",            offsetof(Cookie, value),                0,
    "comment",  offsetof(Cookie, comment),      1,
    "version",          offsetof(Cookie, version),              1,
};
```

⟨*global* `itab` *(webfs)* 61c⟩≡ (80c)

```
struct {
    char *s;
    int offset;
} itab[] = {
    "expire",                   offsetof(Cookie, expire),
    "secure",                   offsetof(Cookie, secure),
    "explicitdomain",   offsetof(Cookie, explicitdom),
    "explicitpath",             offsetof(Cookie, explicitpath),
    "netscapestyle",    offsetof(Cookie, netscapestyle),
};
```

⟨*struct* `Aux` *(webfs)* 61d⟩≡ (80c)

```
struct Aux
{
    char *dom;
    char *path;
    char *inhttp;
    char *outhttp;
    char *ctext;
    int rdoff;
};
```

⟨*function* `jarfmt` *(webfs)* 61e⟩≡ (80c)

```
/* HTTP format */
static int
jarfmt(Fmt *fp)
```

```
{
    int i;
    Jar *jar;

    jar = va_arg(fp->args, Jar*);

    if(jar == nil || jar->nc == 0)
        return 0;

    fmtstrcpy(fp, "Cookie: ");
    if(jar->c[0].version)
        fmtprint(fp, "$Version=%s; ", jar->c[0].version);
    for(i=0; i<jar->nc; i++)
        fmtprint(fp, "%s%s=%s", i ? "; ": "", jar->c[i].name, jar->c[i].value);
    fmtstrcpy(fp, "\r\n");
    return 0;
}
```

⟨*function* cookiefmt*(webfs)* 62a⟩≡ (80c)

```
/* individual cookie */
static int
cookiefmt(Fmt *fp)
{
    int j, k, first;
    char *t;
    Cookie *c;

    c = va_arg(fp->args, Cookie*);

    first = 1;
    for(j=0; j<nelem(stab); j++){
        t = *(char**)((uintptr)c+stab[j].offset);
        if(t == nil)
            continue;
        if(first)
            first = 0;
        else
            fmtstrcpy(fp, " ");
        fmtprint(fp, "%s=%q", stab[j].s, t);
    }
    for(j=0; j<nelem(itab); j++){
        k = *(int*)((uintptr)c+itab[j].offset);
        if(k == 0)
            continue;
        if(first)
            first = 0;
        else
            fmtstrcpy(fp, " ");
        fmtprint(fp, "%s=%ud", itab[j].s, k);
    }
    return 0;
}
```

⟨*function* cookiecmp*(webfs)* 62b⟩≡ (80c)

```
/*
 * sort cookies:
 *      - alpha by name
 *      - alpha by domain
 *      - longer paths first, then alpha by path (RFC2109 4.3.4)
 */
```

```
static int
cookiecmp(Cookie *a, Cookie *b)
{
    int i;

    if((i = strcmp(a->name, b->name)) != 0)
        return i;
    if((i = cistrcmp(a->dom, b->dom)) != 0)
        return i;
    if((i = strlen(b->path) - strlen(a->path)) != 0)
        return i;
    if((i = strcmp(a->path, b->path)) != 0)
        return i;
    return 0;
}
```

⟨*function* `exactcookiecmp` *(webfs)* 63a⟩≡ (80c)

```
static int
exactcookiecmp(Cookie *a, Cookie *b)
{
    int i;

    if((i = cookiecmp(a, b)) != 0)
        return i;
    if((i = strcmp(a->value, b->value)) != 0)
        return i;
    if(a->version || b->version){
        if(!a->version)
            return -1;
        if(!b->version)
            return 1;
        if((i = strcmp(a->version, b->version)) != 0)
            return i;
    }
    if(a->comment || b->comment){
        if(!a->comment)
            return -1;
        if(!b->comment)
            return 1;
        if((i = strcmp(a->comment, b->comment)) != 0)
            return i;
    }
    if((i = b->expire - a->expire) != 0)
        return i;
    if((i = b->secure - a->secure) != 0)
        return i;
    if((i = b->explicitdom - a->explicitdom) != 0)
        return i;
    if((i = b->explicitpath - a->explicitpath) != 0)
        return i;
    if((i = b->netscapestyle - a->netscapestyle) != 0)
        return i;

    return 0;
}
```

⟨*function* `freecookie` *(webfs)* 63b⟩≡ (80c)

```
static void
freecookie(Cookie *c)
{
```

```
    int i;

    for(i=0; i<nelem(stab); i++)
        free(*(char**)((uintptr)c+stab[i].offset));
}
```

⟨*function* `copycookie` *(webfs)* 64a⟩≡ (80c)

```
static void
copycookie(Cookie *c)
{
    int i;
    char **ps;

    for(i=0; i<nelem(stab); i++){
        ps = (char**)((uintptr)c+stab[i].offset);
        if(*ps)
            *ps = estrdup9p(*ps);
    }
}
```

⟨*function* `delcookie` *(webfs)* 64b⟩≡ (80c)

```
static void
delcookie(Jar *j, Cookie *c)
{
    int i;

    j->dirty = 1;
    i = c - j->c;
    if(i < 0 || i >= j->nc)
        abort();
    c->deleted = 1;
}
```

⟨*function* `addcookie` *(webfs)* 64c⟩≡ (80c)

```
static void
addcookie(Jar *j, Cookie *c)
{
    int i;

    if(!c->name || !c->value || !c->path || !c->dom){
        fprint(2, "not adding incomplete cookie\n");
        return;
    }

    if(cookiedebug)
        fprint(2, "add %K\n", c);

    for(i=0; i<j->nc; i++)
        if(cookiecmp(&j->c[i], c) == 0){
            if(cookiedebug)
                fprint(2, "cookie %K matches %K\n", &j->c[i], c);
            if(exactcookiecmp(&j->c[i], c) == 0){
                if(cookiedebug)
                    fprint(2, "\texactly\n");
                j->c[i].mark = 0;
                return;
            }
            delcookie(j, &j->c[i]);
        }
```

```
    j->dirty = 1;
    if(j->nc == j->mc){
        j->mc += 16;
        j->c = erealloc9p(j->c, j->mc*sizeof(Cookie));
    }
    j->c[j->nc] = *c;
    copycookie(&j->c[j->nc]);
    j->nc++;
}
```

⟨*function* `purgejar` *(webfs)* 65a⟩≡ (80c)

```
static void
purgejar(Jar *j)
{
    int i;

    for(i=j->nc-1; i>=0; i--){
        if(!j->c[i].deleted)
            continue;
        freecookie(&j->c[i]);
        --j->nc;
        j->c[i] = j->c[j->nc];
    }
}
```

⟨*function* `addtojar` *(webfs)* 65b⟩≡ (80c)

```
static void
addtojar(Jar *jar, char *line, int ondisk)
{
    Cookie c;
    int i, j, nf, *pint;
    char *f[20], *attr, *val, **pstr;

    memset(&c, 0, sizeof c);
    c.expire = ~0;
    c.ondisk = ondisk;
    nf = tokenize(line, f, nelem(f));
    for(i=0; i<nf; i++){
        attr = f[i];
        if((val = strchr(attr, '=')) != nil)
            *val++ = '\0';
        else
            val = "";
        /* string attributes */
        for(j=0; j<nelem(stab); j++){
            if(strcmp(stab[j].s, attr) == 0){
                pstr = (char**)((uintptr)&c+stab[j].offset);
                *pstr = val;
            }
        }
        /* integer attributes */
        for(j=0; j<nelem(itab); j++){
            if(strcmp(itab[j].s, attr) == 0){
                pint = (int*)((uintptr)&c+itab[j].offset);
                if(val[0]=='\0')
                    *pint = 1;
                else
                    *pint = strtoul(val, 0, 0);
            }
        }
```

```
    }
    if(c.name==nil || c.value==nil || c.dom==nil || c.path==nil){
        if(cookiedebug)
            fprint(2, "ignoring fractional cookie %K\n", &c);
        return;
    }
    addcookie(jar, &c);
}
```

⟨*function* `newjar` *(webfs)* 66a⟩≡ (80c)

```
static Jar*
newjar(void)
{
    Jar *jar;

    jar = emalloc9p(sizeof(Jar));
    return jar;
}
```

⟨*function* `expirejar` *(webfs)* 66b⟩≡ (80c)

```
static int
expirejar(Jar *jar, int exiting)
{
    int i, n;
    uint now;

    now = time(0);
    n = 0;
    for(i=0; i<jar->nc; i++){
        if(jar->c[i].expire < now || (exiting && jar->c[i].expire==~0)){
            delcookie(jar, &jar->c[i]);
            n++;
        }
    }
    return n;
}
```

⟨*function* `dumpjar` *(webfs)* 66c⟩≡ (80c)

```
static void
dumpjar(Jar *jar, char *desc)
{
    int i;
    Biobuf *b;
    char *s;

    print("%s\n", desc);
    print("\tin memory:\n");

    for(i=0; i<jar->nc; i++)
        print("\t%K%s%s%s\n", &jar->c[i],
            jar->c[i].ondisk ? " ondisk" : "",
            jar->c[i].deleted ? " deleted" : "",
            jar->c[i].mark ? " mark" : "");
    print("\n\ton disk:\n");
    if((b = Bopen(jar->file, OREAD)) == nil){
        print("\tno file\n");
    }else{
        while((s = Brdstr(b, '\n', 1)) != nil){
            print("\t%s\n", s);
            free(s);
```

```
        }
        Bterm(b);
    }
    print("\n");
}
```

⟨*function* syncjar*(webfs)* 67⟩≡ (80c)

```
static int
syncjar(Jar *jar)
{
    int i, fd;
    char *line;
    Dir *d;
    Biobuf *b;
    Qid q;

    if(jar->file==nil)
        return 0;

    memset(&q, 0, sizeof q);
    if((d = dirstat(jar->file)) != nil){
        q = d->qid;
        if(d->qid.path != jar->qid.path || d->qid.vers != jar->qid.vers)
            jar->dirty = 1;
        free(d);
    }

    if(jar->dirty == 0)
        return 0;

    fd = -1;
    for(i=0; i<50; i++){
        if((fd = create(jar->lockfile, OWRITE, DMEXCL|0666)) < 0){
            sleep(100);
            continue;
        }
        break;
    }
    if(fd < 0){
        if(cookiedebug)
            fprint(2, "open %s: %r", jar->lockfile);
        werrstr("cannot acquire jar lock: %r");
        return -1;
    }

    for(i=0; i<jar->nc; i++)    /* mark is cleared by addcookie */
        jar->c[i].mark = jar->c[i].ondisk;

    if((b = Bopen(jar->file, OREAD)) == nil){
        if(cookiedebug)
            fprint(2, "Bopen %s: %r", jar->file);
        werrstr("cannot read cookie file %s: %r", jar->file);
        close(fd);
        return -1;
    }
    for(; (line = Brdstr(b, '\n', 1)) != nil; free(line)){
        if(*line == '#')
            continue;
        addtojar(jar, line, 1);
    }
```

```
    Bterm(b);

    for(i=0; i<jar->nc; i++)
        if(jar->c[i].mark && jar->c[i].expire != ~0)
            delcookie(jar, &jar->c[i]);

    purgejar(jar);

    b = Bopen(jar->file, OWRITE);
    if(b == nil){
        if(cookiedebug)
            fprint(2, "Bopen write %s: %r", jar->file);
        close(fd);
        return -1;
    }
    Bprint(b, "# webcookies cookie jar\n");
    Bprint(b, "# comments and non-standard fields will be lost\n");
    for(i=0; i<jar->nc; i++){
        if(jar->c[i].expire == ~0)
            continue;
        Bprint(b, "%K\n", &jar->c[i]);
        jar->c[i].ondisk = 1;
    }
    Bterm(b);

    jar->dirty = 0;
    close(fd);
    if((d = dirstat(jar->file)) != nil){
        jar->qid = d->qid;
        free(d);
    }
    return 0;
}
```

⟨*function* `readjar` *(webfs)* 68⟩≡ (80c)

```
static Jar*
readjar(char *file)
{
    char *lock, *p;
    Jar *jar;

    jar = newjar();
    lock = emalloc9p(strlen(file)+10);
    strcpy(lock, file);
    if((p = strrchr(lock, '/')) != nil)
        p++;
    else
        p = lock;
    memmove(p+2, p, strlen(p)+1);
    p[0] = 'L';
    p[1] = '.';
    jar->lockfile = lock;
    jar->file = file;
    jar->dirty = 1;

    if(syncjar(jar) < 0){
        free(jar->file);
        free(jar->lockfile);
        free(jar);
        return nil;
```

```
    }
    return jar;
}
```

⟨*function* `closejar` *(webfs)* 69a⟩≡ (80c)

```
static void
closejar(Jar *jar)
{
    int i;

    if(jar == nil)
        return;
    expirejar(jar, 0);
    if(syncjar(jar) < 0)
        fprint(2, "warning: cannot rewrite cookie jar: %r\n");

    for(i=0; i<jar->nc; i++)
        freecookie(&jar->c[i]);

    free(jar->file);
    free(jar);
}
```

⟨*function* `isdomainmatch` *(webfs)* 69b⟩≡ (80c)

```
/*
 * Domain name matching is per RFC2109, section 2:
 *
 * Hosts names can be specified either as an IP address or a FQHN
 * string.  Sometimes we compare one host name with another.  Host A's
 * name domain-matches host B's if
 *
 * * both host names are IP addresses and their host name strings match
 *   exactly; or
 *
 * * both host names are FQDN strings and their host name strings match
 *   exactly; or
 *
 * * A is a FQDN string and has the form NB, where N is a non-empty name
 *   string, B has the form .B', and B' is a FQDN string.  (So, x.y.com
 *   domain-matches .y.com but not y.com.)
 *
 * Note that domain-match is not a commutative operation: a.b.c.com
 * domain-matches .c.com, but not the reverse.
 *
 * (This does not verify that IP addresses and FQDN's are well-formed.)
 */
static int
isdomainmatch(char *name, char *pattern)
{
    int lname, lpattern;

    if(cistrcmp(name, pattern)==0)
        return 1;

    if(strcmp(ipattr(name), "dom")==0 && pattern[0]=='.'){
        lname = strlen(name);
        lpattern = strlen(pattern);
        /* e.g., name: www.google.com && pattern: .google.com */
        if(lname >= lpattern && cistrcmp(name+lname-lpattern, pattern)==0)
            return 1;
```

```
        /* e.g., name: google.com && pattern: .google.com */
        if(lpattern > lname &&
            cistrcmp(pattern+lpattern-lname, name) == 0)
            return 1;
    }
    return 0;
}
```

⟨*function* `iscookiematch`*(webfs)* 70a⟩≡ (80c)

```
/*
 * RFC2109 4.3.4:
 *      - domain must match
 *      - path in cookie must be a prefix of request path
 *      - cookie must not have expired
 */
static int
iscookiematch(Cookie *c, char *dom, char *path, uint now)
{
    return isdomainmatch(dom, c->dom)
        && strncmp(c->path, path, strlen(c->path))==0
        && (c->expire == 0 || c->expire >= now);
}
```

⟨*function* `cookiesearch`*(webfs)* 70b⟩≡ (80c)

```
/*
 * Produce a subjar of matching cookies.
 * Secure cookies are only included if secure is set.
 */
static Jar*
cookiesearch(Jar *jar, char *dom, char *path, int issecure)
{
    int i;
    Jar *j;
    uint now;

    if(cookiedebug)
        fprint(2, "cookiesearch %s %s %d\n", dom, path, issecure);
    now = time(0);
    j = newjar();
    for(i=0; i<jar->nc; i++){
        if(cookiedebug)
            fprint(2, "\ttry %s %s %d %s\n", jar->c[i].dom,
                jar->c[i].path, jar->c[i].secure,
                jar->c[i].name);
        if((issecure || !jar->c[i].secure) &&
            iscookiematch(&jar->c[i], dom, path, now)){
            if(cookiedebug)
                fprint(2, "\tmatched\n");
            addcookie(j, &jar->c[i]);
        }
    }
    if(j->nc == 0){
        closejar(j);
        werrstr("no cookies found");
        return nil;
    }
    qsort(j->c, j->nc, sizeof(j->c[0]), (int(*)(void*, void*))cookiecmp);
    return j;
}
```

⟨*function* `isbadcookie` *(webfs)* 71a⟩≡ (80c)

```
/*
 * RFC2109 4.3.2 security checks
 */
static char*
isbadcookie(Cookie *c, char *dom, char *path)
{
    int lcdom, ldom;

    if(strncmp(c->path, path, strlen(c->path)) != 0)
        return "cookie path is not a prefix of the request path";

    /*
     * fgb says omitting this test is necessary to get some sites to work,
     * but it seems dubious.
     */
    if(c->explicitdom && c->dom[0] != '.')
        return "cookie domain doesn't start with dot";

    lcdom = strlen(c->dom);
    if(memchr(c->dom+1, '.', lcdom-1-1) == nil)
        return "cookie domain doesn't have embedded dots";

    if(!isdomainmatch(dom, c->dom))
        return "request host does not match cookie domain";

    ldom = strlen(dom);
    if(strcmp(ipattr(dom), "dom")==0 && lcdom > ldom &&
        memchr(dom, '.', lcdom - ldom) != nil)
        return "request host contains dots before cookie domain";

    return 0;
}
```

⟨*function* `isleap` *(webfs)* 71b⟩≡ (80c)

```
/*
 * Sunday, 25-Jan-2002 12:24:36 GMT
 * Sunday, 25 Jan 2002 12:24:36 GMT
 * Sun, 25 Jan 02 12:24:36 GMT
 */
static int
isleap(int year)
{
    return year%4==0 && (year%100!=0 || year%400==0);
}
```

⟨*function* `strtotime` *(webfs)* 71c⟩≡ (80c)

```
static uint
strtotime(char *s)
{
    char *os;
    int i;
    Tm tm;

    static int mday[2][12] = {
        31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,
        31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,
    };
    static char *wday[] = {
        "Sunday", "Monday", "Tuesday", "Wednesday",
```

```
    "Thursday", "Friday", "Saturday",
};
static char *mon[] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",
};

os = s;
/* Sunday, */
for(i=0; i<nelem(wday); i++){
    if(cistrncmp(s, wday[i], strlen(wday[i])) == 0){
        s += strlen(wday[i]);
        break;
    }
    if(cistrncmp(s, wday[i], 3) == 0){
        s += 3;
        break;
    }
}
if(i==nelem(wday)){
    if(cookiedebug)
        fprint(2, "bad wday (%s)\n", os);
    return -1;
}
if(*s++ != ',' || *s++ != ' '){
    if(cookiedebug)
        fprint(2, "bad wday separator (%s)\n", os);
    return -1;
}

/* 25- */
if(!isdigit(s[0]) || !isdigit(s[1]) || (s[2]!='-' && s[2]!=' ')){
    if(cookiedebug)
        fprint(2, "bad day of month (%s)\n", os);
    return -1;
}
tm.mday = strtol(s, 0, 10);
s += 3;

/* Jan- */
for(i=0; i<nelem(mon); i++)
    if(cistrncmp(s, mon[i], 3) == 0){
        tm.mon = i;
        s += 3;
        break;
    }
if(i==nelem(mon)){
    if(cookiedebug)
        fprint(2, "bad month (%s)\n", os);
    return -1;
}
if(s[0] != '-' && s[0] != ' '){
    if(cookiedebug)
        fprint(2, "bad month separator (%s)\n", os);
    return -1;
}
s++;

/* 2002 */
if(!isdigit(s[0]) || !isdigit(s[1])){
```

```
        if(cookiedebug)
            fprint(2, "bad year (%s)\n", os);
        return -1;
    }
    tm.year = strtol(s, 0, 10);
    s += 2;
    if(isdigit(s[0]) && isdigit(s[1]))
        s += 2;
    else{
        if(tm.year <= 68)
            tm.year += 2000;
        else
            tm.year += 1900;
    }
    if(tm.mday==0 || tm.mday > mday[isleap(tm.year)][tm.mon]){
        if(cookiedebug)
            fprint(2, "invalid day of month (%s)\n", os);
        return -1;
    }
    tm.year -= 1900;
    if(*s++ != ' '){
        if(cookiedebug)
            fprint(2, "bad year separator (%s)\n", os);
        return -1;
    }

    if(!isdigit(s[0]) || !isdigit(s[1]) || s[2]!=':'
    || !isdigit(s[3]) || !isdigit(s[4]) || s[5]!=':'
    || !isdigit(s[6]) || !isdigit(s[7]) || s[8]!=' '){
        if(cookiedebug)
            fprint(2, "bad time (%s)\n", os);
        return -1;
    }

    tm.hour = atoi(s);
    tm.min = atoi(s+3);
    tm.sec = atoi(s+6);
    if(tm.hour >= 24 || tm.min >= 60 || tm.sec >= 60){
        if(cookiedebug)
            fprint(2, "invalid time (%s)\n", os);
        return -1;
    }
    s += 9;

    if(cistrcmp(s, "GMT") != 0){
        if(cookiedebug)
            fprint(2, "time zone not GMT (%s)\n", os);
        return -1;
    }
    strcpy(tm.zone, "GMT");
    tm.yday = 0;
    return tm2sec(&tm);
}
```

⟨*function* `skipspace` *(webfs)* 73⟩≡ (80c)

```
/*
 * skip linear whitespace.  we're a bit more lenient than RFC2616 2.2.
 */
static char*
skipspace(char *s)
```

```
{
    while(*s==’\r’ || *s==’\n’ || *s==’ ’ || *s==’\t’)
        s++;
    return s;
}
```

⟨*function* isnetscape*(webfs)* 74a⟩≡ (80c)

```
/*
 * Try to identify old netscape headers.
 * The old headers:
 *      - didn’t allow spaces around the ’=’
 *      - used an ’Expires’ attribute
 *      - had no ’Version’ attribute
 *      - had no quotes
 *      - allowed whitespace in values
 *      - apparently separated attr/value pairs with ’;’ exclusively
 */
static int
isnetscape(char *hdr)
{
    char *s;

    for(s=hdr; (s=strchr(s, ’=’)) != nil; s++){
        if(isspace(s[1]) || (s > hdr && isspace(s[-1])))
            return 0;
        if(s[1]==’"’)
            return 0;
    }
    if(cistrstr(hdr, "version="))
        return 0;
    return 1;
}
```

⟨*function* parsehttp*(webfs)* 74b⟩≡ (80c)

```
static int
parsehttp(Jar *jar, char *hdr, char *dom, char *path)
{
    static char setcookie[] = "Set-Cookie:";
    char *e, *p, *nextp;
    Cookie c;
    int isns, n;

    isns = isnetscape(hdr);
    n = 0;
    for(p=hdr; p; p=nextp){
        p = skipspace(p);
        if(*p == ’\0’)
            break;
        nextp = strchr(p, ’\n’);
        if(nextp != nil)
            *nextp++ = ’\0’;
        if(cistrncmp(p, setcookie, strlen(setcookie)) != 0)
            continue;
        if(cookiedebug)
            fprint(2, "%s\n", p);
        p = skipspace(p+strlen(setcookie));
        for(; *p; p=skipspace(p)){
            if((e = parsecookie(&c, p, &p, isns, dom, path)) != nil){
                if(cookiedebug)
                    fprint(2, "parse cookie: %s\n", e);
```

```
                break;
            }
            if((e = isbadcookie(&c, dom, path)) != nil){
                if(cookiedebug)
                    fprint(2, "reject cookie; %s\n", e);
                continue;
            }
            addcookie(jar, &c);
            n++;
        }
    }
    return n;
}
```

⟨*function* skipquoted*(webfs)* 75a⟩≡ (80c)

```
static char*
skipquoted(char *s)
{
    /*
     * Sec 2.2 of RFC2616 defines a "quoted-string" as:
     *
     *  quoted-string  = ( <"> *(qdtext | quoted-pair ) <"> )
     *  qdtext         = <any TEXT except <">>
     *  quoted-pair    = "\" CHAR
     *
     * TEXT is any octet except CTLs, but including LWS;
     * LWS is [CR LF] 1*(SP | HT);
     * CHARs are ASCII octets 0-127;  (NOTE: we reject 0's)
     * CTLs are octets 0-31 and 127;
     */
    if(*s != '"')
        return s;

    for(s++; 32 <= *s && *s < 127 && *s != '"'; s++)
        if(*s == '\\' && *(s+1) != '\0')
            s++;
    return s;
}
```

⟨*function* skiptoken*(webfs)* 75b⟩≡ (80c)

```
static char*
skiptoken(char *s)
{
    /*
     * Sec 2.2 of RFC2616 defines a "token" as
         *  1*<any CHAR except CTLs or separators>;
     * CHARs are ASCII octets 0-127;
     * CTLs are octets 0-31 and 127;
     * separators are "()<>@,;:\/[]?={}", double-quote, SP (32), and HT (9)
     */
    while(32 <= *s && *s < 127 && strchr("()<>@,;:[]?={}\" \t\\", *s)==nil)
        s++;

    return s;
}
```

⟨*function* skipvalue*(webfs)* 75c⟩≡ (80c)

```
static char*
skipvalue(char *s, int isns)
{
```

```
    char *t;

    /*
     * An RFC2109 value is an HTTP token or an HTTP quoted string.
     * Netscape servers ignore the spec and rely on semicolons, apparently.
     */
    if(isns){
        if((t = strchr(s, ';')) == nil)
            t = s+strlen(s);
        return t;
    }
    if(*s == '"')
        return skipquoted(s);
    return skiptoken(s);
}
```

⟨*function* parsecookie*(webfs)* 76⟩≡ (80c)

```
/*
 * RMID=80b186bb64c03c65fab767f8; expires=Monday, 10-Feb-2003 04:44:39 GMT;
 *      path=/; domain=.nytimes.com
 */
static char*
parsecookie(Cookie *c, char *p, char **e, int isns, char *dom, char *path)
{
    int i, done;
    char *t, *u, *attr, *val;

    c->expire = ~0;
    memset(c, 0, sizeof *c);

    /* NAME=VALUE */
    t = skiptoken(p);
    c->name = p;
    p = skipspace(t);
    if(*p != '='){
    Badname:
        return "malformed cookie: no NAME=VALUE";
    }
    *t = '\0';
    p = skipspace(p+1);
    t = skipvalue(p, isns);
    if(*t)
        *t++ = '\0';
    c->value = p;
    p = skipspace(t);
    if(c->name[0]=='\0' || c->value[0]=='\0')
        goto Badname;

    done = 0;
    for(; *p && !done; p=skipspace(p)){
        attr = p;
        t = skiptoken(p);
        u = skipspace(t);
        switch(*u){
        case '\0':
            *t = '\0';
            val = p = u;
            break;
        case ';':
            *t = '\0';
```

```
        val = "";
        p = u+1;
        break;
    case '=':
        *t = '\0';
        val = skipspace(u+1);
        p = skipvalue(val, isns);
        if(*p==',')
            done = 1;
        if(*p)
            *p++ = '\0';
        break;
    case ',':
        if(!isns){
            val = "";
            p = u;
            *p++ = '\0';
            done = 1;
            break;
        }
    default:
        if(cookiedebug)
            fprint(2, "syntax: %s\n", p);
        return "syntax error";
    }
    for(i=0; i<nelem(stab); i++)
        if(stab[i].ishttp && cistrcmp(stab[i].s, attr)==0)
            *(char**)((uintptr)c+stab[i].offset) = val;
    if(cistrcmp(attr, "expires") == 0){
        if(!isns)
            return "non-netscape cookie has Expires tag";
        if(!val[0])
            return "bad expires tag";
        c->expire = strtotime(val);
        if(c->expire == ~0)
            return "cannot parse netscape expires tag";
    }
    if(cistrcmp(attr, "max-age") == 0)
        c->expire = time(0)+atoi(val);
    if(cistrcmp(attr, "secure") == 0)
        c->secure = 1;
}

if(c->dom)
    c->explicitdom = 1;
else
    c->dom = dom;
if(c->path)
    c->explicitpath = 1;
else{
    c->path = path;
    if((t = strchr(c->path, '?')) != 0)
        *t = '\0';
    if((t = strrchr(c->path, '/')) != 0)
        *t = '\0';
}
c->netscapestyle = isns;
*e = p;

return nil;
```

```
}
```

⟨*function* cookieopen*(webfs)* 78a⟩≡ (80c)

```
void
cookieopen(Req *r)
{
    char *s, *es;
    int i, sz;
    Aux *a;

    syncjar(jar);
    a = emalloc9p(sizeof(Aux));
    r->fid->aux = a;
    if(r->ifcall.mode&OTRUNC){
        a->ctext = emalloc9p(1);
        a->ctext[0] = '\0';
    }else{
        sz = 256*jar->nc+1024;  /* BUG should do better */
        a->ctext = emalloc9p(sz);
        a->ctext[0] = '\0';
        s = a->ctext;
        es = s+sz;
        for(i=0; i<jar->nc; i++)
            s = seprint(s, es, "%K\n", &jar->c[i]);
    }
    respond(r, nil);
}
```

⟨*function* cookieread*(webfs)* 78b⟩≡ (80c)

```
void
cookieread(Req *r)
{
    Aux *a;

    a = r->fid->aux;
    readstr(r, a->ctext);
    respond(r, nil);
}
```

⟨*function* cookiewrite*(webfs)* 78c⟩≡ (80c)

```
void
cookiewrite(Req *r)
{
    Aux *a;
    int sz;

    a = r->fid->aux;
    sz = r->ifcall.count+r->ifcall.offset;
    if(sz > strlen(a->ctext)){
        if(sz >= MaxCtext){
            respond(r, "cookie file too large");
            return;
        }
        a->ctext = erealloc9p(a->ctext, sz+1);
        a->ctext[sz] = '\0';
    }
    memmove(a->ctext+r->ifcall.offset, r->ifcall.data, r->ifcall.count);
    r->ofcall.count = r->ifcall.count;
    respond(r, nil);
}
```

⟨*function* `cookieclunk` *(webfs)* 79a⟩≡ (80c)

```
void
cookieclunk(Fid *fid)
{
    char *p, *nextp;
    Aux *a;
    int i;

    a = fid->aux;
    if(a == nil)
        return;
    for(i=0; i<jar->nc; i++)
        jar->c[i].mark = 1;
    for(p=a->ctext; *p; p=nextp){
        if((nextp = strchr(p, '\n')) != nil)
            *nextp++ = '\0';
        else
            nextp = "";
        addtojar(jar, p, 0);
    }
    for(i=0; i<jar->nc; i++)
        if(jar->c[i].mark)
            delcookie(jar, &jar->c[i]);
    syncjar(jar);
    free(a->dom);
    free(a->path);
    free(a->inhttp);
    free(a->outhttp);
    free(a->ctext);
    free(a);
}
```

⟨*function* `closecookies` *(webfs)* 79b⟩≡ (80c)

```
void
closecookies(void)
{
    closejar(jar);
}
```

⟨*function* `initcookies` *(webfs)* 79c⟩≡ (80c)

```
void
initcookies(char *file)
{
    char *home;

    fmtinstall('J', jarfmt);
    fmtinstall('K', cookiefmt);

    if(file == nil){
        home = getenv("home");
        if(home == nil)
            sysfatal("no cookie file specified and no $home");
        file = emalloc9p(strlen(home)+30);
        strcpy(file, home);
        strcat(file, "/lib/webcookies");
    }
    jar = readjar(file);
    if(jar == nil)
        sysfatal("readjar: %r");
}
```

⟨*function* `httpsetcookie` *(webfs)* 80a⟩≡ (80c)

```
void
httpsetcookie(char *hdr, char *dom, char *path)
{
    if(path == nil)
        path = "/";

    parsehttp(jar, hdr, dom, path);
    syncjar(jar);
}
```

⟨*function* `httpcookies` *(webfs)* 80b⟩≡ (80c)

```
char*
httpcookies(char *dom, char *path, int issecure)
{
    char buf[1024];
    Jar *j;

    syncjar(jar);
    j = cookiesearch(jar, dom, path, issecure);
    snprint(buf, sizeof buf, "%J", j);
    closejar(j);
    return estrdup(buf);
}
```

⟨`webfs/cookies.c` 80c⟩≡

```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ndb.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>
#include <ctype.h>
#include "dat.h"
#include "fns.h"

int cookiedebug;

typedef struct Cookie Cookie;
typedef struct Jar Jar;
```

⟨*struct* `Cookie` *(webfs)* 60⟩

⟨*struct* `Jar` *(webfs)* 61a⟩

⟨*global* `stab` *(webfs)* 61b⟩
⟨*global* `itab` *(webfs)* 61c⟩

```
#pragma varargck type "J"       Jar*
#pragma varargck type "K"       Cookie*
```

⟨*function* `jarfmt` *(webfs)* 61e⟩

⟨*function* `cookiefmt` *(webfs)* 62a⟩

⟨*function* `cookiecmp` *(webfs)* 62b⟩

⟨*function* `exactcookiecmp` *(webfs)* 63a⟩

⟨*function* `freecookie` *(webfs)* 63b⟩

⟨*function* `copycookie` *(webfs)* 64a⟩

⟨*function* `delcookie` *(webfs)* 64b⟩

⟨*function* `addcookie` *(webfs)* 64c⟩

⟨*function* `purgejar` *(webfs)* 65a⟩

⟨*function* `addtojar` *(webfs)* 65b⟩

⟨*function* `newjar` *(webfs)* 66a⟩

⟨*function* `expirejar` *(webfs)* 66b⟩

⟨*function* `dumpjar` *(webfs)* 66c⟩

⟨*function* `syncjar` *(webfs)* 67⟩

⟨*function* `readjar` *(webfs)* 68⟩

⟨*function* `closejar` *(webfs)* 69a⟩

⟨*function* `isdomainmatch` *(webfs)* 69b⟩

⟨*function* `iscookiematch` *(webfs)* 70a⟩

⟨*function* `cookiesearch` *(webfs)* 70b⟩

⟨*function* `isbadcookie` *(webfs)* 71a⟩

⟨*function* `isleap` *(webfs)* 71b⟩

⟨*function* `strtotime` *(webfs)* 71c⟩

⟨*function* `skipspace` *(webfs)* 73⟩

⟨*function* `isnetscape` *(webfs)* 74a⟩

```
/*
 * Parse HTTP response headers, adding cookies to jar.
 * Overwrites the headers.  May overwrite path.
 */
static char* parsecookie(Cookie*, char*, char**, int, char*, char*);
```

⟨*function* `parsehttp` *(webfs)* 74b⟩

⟨*function* `skipquoted` *(webfs)* 75a⟩

⟨*function* `skiptoken` *(webfs)* 75b⟩

⟨*function* `skipvalue` *(webfs)* 75c⟩

⟨*function* `parsecookie` *(webfs)* 76⟩

```
Jar *jar;

typedef struct Aux Aux;
```

⟨*struct* `Aux` *(webfs)* 61d⟩

```
enum
{
    AuxBuf = 4096,
    MaxCtext = 16*1024*1024,
};
```

⟨*function* `cookieopen` *(webfs)* 78a⟩

⟨*function* `cookieread` *(webfs)* 78b⟩

⟨*function* `cookiewrite` *(webfs)* 78c⟩

⟨*function* `cookieclunk` *(webfs)* 79a⟩

⟨*function* `closecookies` *(webfs)* 79b⟩

⟨*function* `initcookies` *(webfs)* 79c⟩

⟨*function* `httpsetcookie` *(webfs)* 80a⟩

⟨*function* `httpcookies` *(webfs)* 80b⟩

webfs/fs.c

⟨*enum Qxxx (webfs)* 82a⟩≡ (92c)

```
enum
{
    Qroot,
    Qrootctl,
    Qclone,
    Qcookies,
    Qclient,
    Qctl,
    Qbody,
    Qbodyext,
    Qcontenttype,
    Qpostbody,
    Qparsed,
    Qurl,
    Qscheme,
    Qschemedata,
    Quser,
    Qpasswd,
    Qhost,
    Qport,
    Qpath,
    Qquery,
    Qfragment,
    Qftptype,
    Qend,
};
```

⟨*struct* `Tab` *(webfs)* 82b⟩≡ (92c)

```
struct Tab
{
    char *name;
    ulong mode;
    int offset;
};
```

⟨*global* `tab` *(webfs)* 83a⟩≡ (92c)

```
Tab tab[] =
{
    "/",                        DMDIR|0555,             0,
    "ctl",                      0666,                   0,
    "clone",            0666,                   0,
    "cookies",          0666,                   0,
    "XXX",              DMDIR|0555,             0,
    "ctl",                      0666,                   0,
    "body",             0444,                   0,
    "XXX",              0444,                   0,
    "contenttype",      0444,                   0,
    "postbody", 0666,                   0,
    "parsed",           DMDIR|0555,             0,
    "url",                      0444,                   offsetof(Url, url),
    "scheme",           0444,                   offsetof(Url, scheme),
    "schemedata",       0444,                   offsetof(Url, schemedata),
    "user",             0444,                   offsetof(Url, user),
    "passwd",           0444,                   offsetof(Url, passwd),
    "host",             0444,                   offsetof(Url, host),
    "port",             0444,                   offsetof(Url, port),
    "path",             0444,                   offsetof(Url, path),
    "query",            0444,                   offsetof(Url, query),
    "fragment", 0444,                   offsetof(Url, fragment),
    "ftptype",          0444,                   offsetof(Url, ftp.type),
};
```

⟨*constant* `fs` *(webfs)* 83b⟩≡ (92c)

```
Srv fs =
{
.attach=                fssend,
.destroyfid=    fsdestroyfid,
.walk1=         fswalk1,
.open=          fssend,
.read=          fssend,
.write=         fssend,
.stat=          fssend,
.flush=         fssend,
.end=           takedown,
};
```

⟨*macro* `PATH` *(webfs)* 83c⟩≡ (92c)

```
#define PATH(type, n)   ((type)|((n)<<8))
```

⟨*macro* `TYPE` *(webfs)* 83d⟩≡ (92c)

```
#define TYPE(path)              ((int)(path) & 0xFF)
```

⟨*macro* `NUM` *(webfs)* 83e⟩≡ (92c)

```
#define NUM(path)               ((uint)(path)>>8)
```

⟨*function* `fillstat` *(webfs)* 83f⟩≡ (92c)

```
static void
fillstat(Dir *d, uvlong path, ulong length, char *ext)
{
    Tab *t;
    int type;
    char buf[32];

    memset(d, 0, sizeof(*d));
    d->uid = estrdup("web");
```

```
    d->gid = estrdup("web");
    d->qid.path = path;
    d->atime = d->mtime = time0;
    d->length = length;
    type = TYPE(path);
    t = &tab[type];
    if(type == Qbodyext) {
        snprint(buf, sizeof buf, "body.%s", ext == nil ? "xxx" : ext);
        d->name = estrdup(buf);
    }
    else if(t->name)
        d->name = estrdup(t->name);
    else{       /* client directory */
        snprint(buf, sizeof buf, "%ud", NUM(path));
        d->name = estrdup(buf);
    }
    d->qid.type = t->mode>>24;
    d->mode = t->mode;
}
```

⟨*function* `fsstat` *(webfs)* 84a⟩≡ (92c)

```
static void
fsstat(Req *r)
{
    fillstat(&r->d, r->fid->qid.path, 0, nil);
    respond(r, nil);
}
```

⟨*function* `rootgen` *(webfs)* 84b⟩≡ (92c)

```
static int
rootgen(int i, Dir *d, void*)
{
    char buf[32];

    i += Qroot+1;
    if(i < Qclient){
        fillstat(d, i, 0, nil);
        return 0;
    }
    i -= Qclient;
    if(i < nclient){
        fillstat(d, PATH(Qclient, i), 0, nil);
        snprint(buf, sizeof buf, "%d", i);
        free(d->name);
        d->name = estrdup(buf);
        return 0;
    }
    return -1;
}
```

⟨*function* `clientgen` *(webfs)* 84c⟩≡ (92c)

```
static int
clientgen(int i, Dir *d, void *aux)
{
    Client *c;

    c = aux;
    i += Qclient+1;
    if(i <= Qparsed){
        fillstat(d, PATH(i, c->num), 0, c->ext);
```

```
        return 0;
    }
    return -1;
}
```

⟨*function* `parsedgen` *(webfs)* 85a⟩≡ (92c)

```
static int
parsedgen(int i, Dir *d, void *aux)
{
    Client *c;

    c = aux;
    i += Qparsed+1;
    if(i < Qend){
        fillstat(d, PATH(i, c->num), 0, nil);
        return 0;
    }
    return -1;
}
```

⟨*function* `fsread` *(webfs)* 85b⟩≡ (92c)

```
static void
fsread(Req *r)
{
    char *s;
    char e[ERRMAX];
    Client *c;
    ulong path;

    path = r->fid->qid.path;
    switch(TYPE(path)){
    default:
        snprint(e, sizeof e, "bug in webfs path=%lux\n", path);
        respond(r, e);
        break;

    case Qroot:
        dirread9p(r, rootgen, nil);
        respond(r, nil);
        break;

    case Qrootctl:
        globalctlread(r);
        break;

    case Qcookies:
        cookieread(r);
        break;

    case Qclient:
        dirread9p(r, clientgen, client[NUM(path)]);
        respond(r, nil);
        break;

    case Qctl:
        ctlread(r, client[NUM(path)]);
        break;

    case Qcontenttype:
        c = client[NUM(path)];
```

```
        if(c->contenttype == nil)
            r->ofcall.count = 0;
        else
            readstr(r, c->contenttype);
        respond(r, nil);
        break;

    case Qpostbody:
        c = client[NUM(path)];
        readbuf(r, c->postbody, c->npostbody);
        respond(r, nil);
        break;

    case Qbody:
    case Qbodyext:
        c = client[NUM(path)];
        if(c->iobusy){
            respond(r, "already have i/o pending");
            break;
        }
        c->iobusy = 1;
        sendp(c->creq, r);
        break;

    case Qparsed:
        dirread9p(r, parsedgen, client[NUM(path)]);
        respond(r, nil);
        break;

    case Qurl:
    case Qscheme:
    case Qschemedata:
    case Quser:
    case Qpasswd:
    case Qhost:
    case Qport:
    case Qpath:
    case Qquery:
    case Qfragment:
    case Qftptype:
        c = client[NUM(path)];
        r->ofcall.count = 0;
        if(c->url != nil
        && (s = *(char**)((uintptr)c->url+tab[TYPE(path)].offset)) != nil)
            readstr(r, s);
        respond(r, nil);
        break;
    }
}
```

⟨*function* `fswrite` *(webfs)* 86⟩≡ (92c)

```
static void
fswrite(Req *r)
{
    int m;
    ulong path;
    char e[ERRMAX], *buf, *cmd, *arg;
    Client *c;

    path = r->fid->qid.path;
```

```
    switch(TYPE(path)){
    default:
        snprint(e, sizeof e, "bug in webfs path=%lux\n", path);
        respond(r, e);
        break;

    case Qcookies:
        cookiewrite(r);
        break;

    case Qrootctl:
    case Qctl:
        if(r->ifcall.count >= 1024){
            respond(r, "ctl message too long");
            return;
        }
        buf = estredup(r->ifcall.data, (char*)r->ifcall.data+r->ifcall.count);
        cmd = buf;
        arg = strpbrk(cmd, "\t ");
        if(arg){
            *arg++ = '\0';
            arg += strspn(arg, "\t ");
        }else
            arg = "";
        r->ofcall.count = r->ifcall.count;
        if(TYPE(path)==Qrootctl){
            if(!ctlwrite(r, &globalctl, cmd, arg)
            && !globalctlwrite(r, cmd, arg))
                respond(r, "unknown control command");
        }else{
            c = client[NUM(path)];
            if(!ctlwrite(r, &c->ctl, cmd, arg)
            && !clientctlwrite(r, c, cmd, arg))
                respond(r, "unknown control command");
        }
        free(buf);
        break;

    case Qpostbody:
        c = client[NUM(path)];
        if(c->bodyopened){
            respond(r, "cannot write postbody after opening body");
            break;
        }
        if(r->ifcall.offset >= 128*1024*1024){  /* >128MB is probably a mistake */
            respond(r, "offset too large");
            break;
        }
        m = r->ifcall.offset + r->ifcall.count;
        if(c->npostbody < m){
            c->postbody = erealloc(c->postbody, m);
            memset(c->postbody+c->npostbody, 0, m-c->npostbody);
            c->npostbody = m;
        }
        memmove(c->postbody+r->ifcall.offset, r->ifcall.data, r->ifcall.count);
        r->ofcall.count = r->ifcall.count;
        respond(r, nil);
        break;
    }
}
```

⟨*function* `fsopen`*(webfs)* 88⟩≡ (92c)

```
static void
fsopen(Req *r)
{
    static int need[4] = { 4, 2, 6, 1 };
    ulong path;
    int n;
    Client *c;
    Tab *t;

    /*
     * lib9p already handles the blatantly obvious.
     * we just have to enforce the permissions we have set.
     */
    path = r->fid->qid.path;
    t = &tab[TYPE(path)];
    n = need[r->ifcall.mode&3];
    if((n&t->mode) != n){
        respond(r, "permission denied");
        return;
    }

    switch(TYPE(path)){
    case Qcookies:
        cookieopen(r);
        break;

    case Qpostbody:
        c = client[NUM(path)];
        c->havepostbody++;
        c->ref++;
        respond(r, nil);
        break;

    case Qbody:
    case Qbodyext:
        c = client[NUM(path)];
        if(c->url == nil){
            respond(r, "url is not yet set");
            break;
        }
        c->bodyopened = 1;
        c->ref++;
        sendp(c->creq, r);
        break;

    case Qclone:
        n = newclient(0);
        path = PATH(Qctl, n);
        r->fid->qid.path = path;
        r->ofcall.qid.path = path;
        if(fsdebug)
            fprint(2, "open clone => path=%lux\n", path);
        t = &tab[Qctl];
        /* fall through */
    default:
        if(t-tab >= Qclient)
            client[NUM(path)]->ref++;
        respond(r, nil);
        break;
```

```
    }
}
```

⟨*function* `fsdestroyfid` *(webfs)* 89a⟩≡ (92c)

```
static void
fsdestroyfid(Fid *fid)
{
    sendp(cclunk, fid);
    recvp(cclunkwait);
}
```

⟨*function* `fsattach` *(webfs)* 89b⟩≡ (92c)

```
static void
fsattach(Req *r)
{
    if(r->ifcall.aname && r->ifcall.aname[0]){
        respond(r, "invalid attach specifier");
        return;
    }
    r->fid->qid.path = PATH(Qroot, 0);
    r->fid->qid.type = QTDIR;
    r->fid->qid.vers = 0;
    r->ofcall.qid = r->fid->qid;
    respond(r, nil);
}
```

⟨*function* `fswalk1` *(webfs)* 89c⟩≡ (92c)

```
static char*
fswalk1(Fid *fid, char *name, Qid *qid)
{
    int i, n;
    ulong path;
    char buf[32], *ext;

    path = fid->qid.path;
    if(!(fid->qid.type&QTDIR))
        return "walk in non-directory";

    if(strcmp(name, "..") == 0){
        switch(TYPE(path)){
        case Qparsed:
            qid->path = PATH(Qclient, NUM(path));
            qid->type = tab[Qclient].mode>>24;
            return nil;
        case Qclient:
        case Qroot:
            qid->path = PATH(Qroot, 0);
            qid->type = tab[Qroot].mode>>24;
            return nil;
        default:
            return "bug in fswalk1";
        }
    }

    i = TYPE(path)+1;
    for(; i<nelem(tab); i++){
        if(i==Qclient){
            n = atoi(name);
            snprint(buf, sizeof buf, "%d", n);
            if(n < nclient && strcmp(buf, name) == 0){
```

```
                qid->path = PATH(i, n);
                qid->type = tab[i].mode>>24;
                return nil;
            }
            break;
        }
        if(i==Qbodyext){
            ext = client[NUM(path)]->ext;
            snprint(buf, sizeof buf, "body.%s", ext == nil ? "xxx" : ext);
            if(strcmp(buf, name) == 0){
                qid->path = PATH(i, NUM(path));
                qid->type = tab[i].mode>>24;
                return nil;
            }
        }
        else if(strcmp(name, tab[i].name) == 0){
            qid->path = PATH(i, NUM(path));
            qid->type = tab[i].mode>>24;
            return nil;
        }
        if(tab[i].mode&DMDIR)
            break;
    }
    return "directory entry not found";
}
```

⟨*function* `fsflush`*(webfs)* 90a⟩≡ (92c)

```
static void
fsflush(Req *r)
{
    Req *or;
    int t;
    Client *c;
    ulong path;

    or=r;
    while(or->ifcall.type==Tflush)
        or = or->oldreq;

    if(or->ifcall.type != Tread && or->ifcall.type != Topen)
        abort();

    path = or->fid->qid.path;
    t = TYPE(path);
    if(t != Qbody && t != Qbodyext)
        abort();

    c = client[NUM(path)];
    sendp(c->creq, r);
    iointerrupt(c->io);
}
```

⟨*function* `fsthread`*(webfs)* 90b⟩≡ (92c)

```
static void
fsthread(void*)
{
    ulong path;
    Alt a[3];
    Fid *fid;
    Req *r;
```

```
    threadsetname("fsthread");
    plumbstart();

    a[0].op = CHANRCV;
    a[0].c = cclunk;
    a[0].v = &fid;
    a[1].op = CHANRCV;
    a[1].c = creq;
    a[1].v = &r;
    a[2].op = CHANEND;

    for(;;){
        switch(alt(a)){
        case 0:
            path = fid->qid.path;
            if(TYPE(path)==Qcookies)
                cookieclunk(fid);
            if(fid->omode != -1 && TYPE(path) >= Qclient)
                closeclient(client[NUM(path)]);
            sendp(cclunkwait, nil);
            break;
        case 1:
            switch(r->ifcall.type){
            case Tattach:
                fsattach(r);
                break;
            case Topen:
                fsopen(r);
                break;
            case Tread:
                fsread(r);
                break;
            case Twrite:
                fswrite(r);
                break;
            case Tstat:
                fsstat(r);
                break;
            case Tflush:
                fsflush(r);
                break;
            default:
                respond(r, "bug in fsthread");
                break;
            }
            sendp(creqwait, 0);
            break;
        }
    }
}
```

⟨*function* `fssend` *(webfs)* 91⟩≡ (92c)

```
static void
fssend(Req *r)
{
    sendp(creq, r);
    recvp(creqwait);    /* avoids need to deal with spurious flushes */
}
```

⟨*function* `initfs`*(webfs)* 92a⟩≡ (92c)

```
void
initfs(void)
{
    time0 = time(0);
    creq = chancreate(sizeof(void*), 0);
    creqwait = chancreate(sizeof(void*), 0);
    cclunk = chancreate(sizeof(void*), 0);
    cclunkwait = chancreate(sizeof(void*), 0);
    procrfork(fsthread, nil, STACK, RFNAMEG);
}
```

⟨*function* `takedown`*(webfs)* 92b⟩≡ (92c)

```
void
takedown(Srv*)
{
    closecookies();
    threadexitsall("done");
}
```

⟨`webfs/fs.c` 92c⟩≡

```
/*
 * Web file system.  Conventionally mounted at /mnt/web
 *
 *      ctl                             send control messages (might go away)
 *      cookies                 list of cookies, editable
 *      clone                   open and read to obtain new connection
 *      n                               connection directory
 *              ctl                             control messages (like get url)
 *              body                            retrieved data
 *              content-type            mime content-type of body
 *              postbody                        data to be posted
 *              parsed                  parsed version of url
 *                      url                             entire url
 *                      scheme                  http, ftp, etc.
 *                      host                            hostname
 *                      path                            path on host
 *                      query                   query after path
 *                      fragment                        #foo anchor reference
 *                      user                            user name (ftp)
 *                      password                        password (ftp)
 *                      ftptype                 transfer mode (ftp)
 */

#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ip.h>
#include <plumb.h>
#include <thread.h>
#include <fcall.h>
#include <9p.h>
#include "dat.h"
#include "fns.h"

int fsdebug;

⟨enum Qxxx(webfs) 82a⟩

⟨macro PATH(webfs) 83c⟩
```

⟨*macro* TYPE*(webfs)* 83d⟩
⟨*macro* NUM*(webfs)* 83e⟩

```
Channel *creq;
Channel *creqwait;
Channel *cclunk;
Channel *cclunkwait;

typedef struct Tab Tab;
```
⟨*struct* Tab*(webfs)* 82b⟩

⟨*global* tab*(webfs)* 83a⟩

```
ulong time0;
```

⟨*function* fillstat*(webfs)* 83f⟩

⟨*function* fsstat*(webfs)* 84a⟩

⟨*function* rootgen*(webfs)* 84b⟩

⟨*function* clientgen*(webfs)* 84c⟩

⟨*function* parsedgen*(webfs)* 85a⟩

⟨*function* fsread*(webfs)* 85b⟩

⟨*function* fswrite*(webfs)* 86⟩

⟨*function* fsopen*(webfs)* 88⟩

⟨*function* fsdestroyfid*(webfs)* 89a⟩

⟨*function* fsattach*(webfs)* 89b⟩

⟨*function* fswalk1*(webfs)* 89c⟩

⟨*function* fsflush*(webfs)* 90a⟩

⟨*function* fsthread*(webfs)* 90b⟩

⟨*function* fssend*(webfs)* 91⟩

⟨*function* initfs*(webfs)* 92a⟩

⟨*function* takedown*(webfs)* 92b⟩

⟨*constant* fs*(webfs)* 83b⟩

webfs/http.c

⟨*struct* HttpState*(webfs)* 93⟩≡ (102c)

```
struct HttpState
{
    int fd;
    Client *c;
    char *location;
    char *setcookie;
```

```
    char *netaddr;
    char *credentials;
    char autherror[ERRMAX];
    Ibuf        b;
};
```

⟨*global* `hdrtab` *(webfs)* 94a⟩≡ (102c)

```
struct {
    char *name;                                          /* Case-insensitive */
    void (*fn)(HttpState *hs, char *value);
} hdrtab[] = {
    { "location:", location },
    { "content-type:", contenttype },
    { "set-cookie:", setcookie },
    { "www-authenticate:", wwwauthenticate },
};
```

⟨*function* `location` *(webfs)* 94b⟩≡ (102c)

```
static void
location(HttpState *hs, char *value)
{
    if(hs->location == nil)
        hs->location = estrdup(value);
}
```

⟨*function* `contenttype` *(webfs)* 94c⟩≡ (102c)

```
static void
contenttype(HttpState *hs, char *value)
{
    if(hs->c->contenttype != nil)
        free(hs->c->contenttype);
    hs->c->contenttype = estrdup(value);
}
```

⟨*function* `setcookie` *(webfs)* 94d⟩≡ (102c)

```
static void
setcookie(HttpState *hs, char *value)
{
    char *s, *t;
    Fmt f;

    s = hs->setcookie;
    fmtstrinit(&f);
    if(s)
        fmtprint(&f, "%s", s);
    fmtprint(&f, "set-cookie: ");
    fmtprint(&f, "%s", value);
    fmtprint(&f, "\n");
    t = fmtstrflush(&f);
    if(t){
        free(s);
        hs->setcookie = t;
    }
}
```

⟨*function* `unquote` *(webfs)* 94e⟩≡ (102c)

```
static char*
unquote(char *s, char **ps)
{
    char *p;
```

```
    if(*s != '"'){
        p = strpbrk(s, " \t\r\n");
        *p++ = 0;
        *ps = p;
        return s;
    }
    for(p=s+1; *p; p++){
        if(*p == '\"'){
            *p++ = 0;
            break;
        }
        if(*p == '\\' && *(p+1)){
            p++;
            continue;
        }
    }
    memmove(s, s+1, p-(s+1));
    s[p-(s+1)] = 0;
    *ps = p;
    return s;
}
```

⟨*function* servername*(webfs)* 95a⟩≡ (102c)

```
static char*
servername(char *addr)
{
    char *p;

    if(strncmp(addr, "tcp!", 4) == 0
    || strncmp(addr, "net!", 4) == 0)
        addr += 4;
    addr = estrdup(addr);
    p = addr+strlen(addr);
    if(p>addr && *(p-1) == 's')
        p--;
    if(p>addr+5 && strcmp(p-5, "!http") == 0)
        p[-5] = 0;
    return addr;
}
```

⟨*function* wwwauthenticate*(webfs)* 95b⟩≡ (102c)

```
void
wwwauthenticate(HttpState *hs, char *line)
{
    char cred[64], *user, *pass, *realm, *s, *spec, *name;
    Fmt fmt;
    UserPasswd *up;

    spec = nil;
    up = nil;
    cred[0] = 0;
    hs->autherror[0] = 0;
    if(cistrncmp(line, "basic ", 6) != 0){
        werrstr("unknown auth: %s", line);
        goto error;
    }
    line += 6;
    if(cistrncmp(line, "realm=", 6) != 0){
        werrstr("missing realm: %s", line);
```

```
        goto error;
    }
    line += 6;
    user = hs->c->url->user;
    pass = hs->c->url->passwd;
    if(user==nil || pass==nil){
        realm = unquote(line, &line);
        fmtstrinit(&fmt);
        name = servername(hs->netaddr);
        fmtprint(&fmt, "proto=pass service=http server=%q realm=%q", name, realm);
        free(name);
        if(hs->c->url->user)
            fmtprint(&fmt, " user=%q", hs->c->url->user);
        spec = fmtstrflush(&fmt);
        if(spec == nil)
            goto error;
        if((up = auth_getuserpasswd(nil, "%s", spec)) == nil)
            goto error;
        user = up->user;
        pass = up->passwd;
    }
    if((s = smprint("%s:%s", user, pass)) == nil)
        goto error;
    free(up);
    enc64(cred, sizeof(cred), (uchar*)s, strlen(s));
    memset(s, 0, strlen(s));
    free(s);
    hs->credentials = smprint("Basic %s", cred);
    if(hs->credentials == nil)
        goto error;
    return;

error:
    free(up);
    free(spec);
    snprint(hs->autherror, sizeof hs->autherror, "%r");
    fprint(2, "%s: Authentication failed: %r\n", argv0);
}
```

⟨*function* `httprcode` *(webfs)* 96⟩≡ (102c)

```
static int
httprcode(HttpState *hs)
{
    int n;
    char *p;
    char buf[256];

    n = readline(&hs->b, buf, sizeof(buf)-1);
    if(n <= 0)
        return n;
    if(httpdebug)
        fprint(2, "-> %s\n", buf);
    p = strchr(buf, ' ');
    if(memcmp(buf, "HTTP/", 5) != 0 || p == nil){
        werrstr("bad response from server");
        return -1;
    }
    buf[n] = 0;
    return atoi(p+1);
}
```

⟨*function* `getheader` *(webfs)* 97a⟩≡ (102c)

```
/*
 *  read a single mime header, collect continuations.
 *
 *  this routine assumes that there is a blank line twixt
 *  the header and the message body, otherwise bytes will
 *  be lost.
 */
static int
getheader(HttpState *hs, char *buf, int n)
{
    char *p, *e;
    int i;

    n--;
    p = buf;
    for(e = p + n; ; p += i){
        i = readline(&hs->b, p, e-p);
        if(i < 0)
            return i;

        if(p == buf){
            /* first line */
            if(strchr(buf, ':') == nil)
                break;          /* end of headers */
        } else {
            /* continuation line */
            if(*p != ' ' && *p != '\t'){
                unreadline(&hs->b, p);
                *p = 0;
                break;          /* end of this header */
            }
        }
    }

    if(httpdebug)
        fprint(2, "-> %s\n", buf);
    return p-buf;
}
```

⟨*function* `httpheaders` *(webfs)* 97b⟩≡ (102c)

```
static int
httpheaders(HttpState *hs)
{
    char buf[2048];
    char *p;
    int i, n;

    for(;;){
        n = getheader(hs, buf, sizeof(buf));
        if(n < 0)
            return -1;
        if(n == 0)
            return 0;
        //      print("http header: '%.*s'\n", n, buf);
        for(i = 0; i < nelem(hdrtab); i++){
            n = strlen(hdrtab[i].name);
            if(cistrncmp(buf, hdrtab[i].name, n) == 0){
                /* skip field name and leading white */
                p = buf + n;
```

```
                while(*p == ' ' || *p == '\t')
                    p++;
                (*hdrtab[i].fn)(hs, p);
                break;
            }
        }
    }
}
```

⟨*function* `httpopen` *(webfs)* 98⟩≡ (102c)

```
int
httpopen(Client *c, Url *url)
{
    int fd, code, redirect, authenticate;
    char *cookies;
    Ioproc *io;
    HttpState *hs;
    char *service;

    if(httpdebug)
        fprint(2, "httpopen\n");
    io = c->io;
    hs = emalloc(sizeof(*hs));
    hs->c = c;

    if(url->port)
        service = url->port;
    else
        service = url->scheme;
    hs->netaddr = estrdup(netmkaddr(url->host, 0, service));
    c->aux = hs;
    if(httpdebug){
        fprint(2, "dial %s\n", hs->netaddr);
        fprint(2, "dial port: %s\n", url->port);
    }
    fd = iotlsdial(io, hs->netaddr, 0, 0, 0, url->ischeme==UShttps, url->host);
    if(fd < 0){
    Error:
        if(httpdebug)
            fprint(2, "iodial: %r\n");
        free(hs->location);
        free(hs->setcookie);
        free(hs->netaddr);
        free(hs->credentials);
        if(fd >= 0)
            ioclose(io, hs->fd);
        hs->fd = -1;
        free(hs);
        c->aux = nil;
        return -1;
    }
    hs->fd = fd;
    if(httpdebug)
        fprint(2, "<- %s %s HTTP/1.0\n<- Host: %s\n",
            c->havepostbody? "POST": "GET", url->http.page_spec, url->host);
    ioprint(io, fd, "%s %s HTTP/1.0\r\nHost: %s\r\n",
        c->havepostbody? "POST" : "GET", url->http.page_spec, url->host);
    if(httpdebug)
        fprint(2, "<- User-Agent: %s\n", c->ctl.useragent);
    if(c->ctl.useragent)
```

```
        ioprint(io, fd, "User-Agent: %s\r\n", c->ctl.useragent);
    if(c->ctl.sendcookies){
        /* should we use url->page here?  sometimes it is nil. */
        cookies = httpcookies(url->host, url->http.page_spec,
            url->ischeme == USHttps);
        if(cookies && cookies[0])
            ioprint(io, fd, "%s", cookies);
        if(httpdebug)
            fprint(2, "<- %s", cookies);
        free(cookies);
    }
    if(c->havepostbody){
        ioprint(io, fd, "Content-type: %s\r\n", PostContentType);
        ioprint(io, fd, "Content-length: %ud\r\n", c->npostbody);
        if(httpdebug){
            fprint(2, "<- Content-type: %s\n", PostContentType);
            fprint(2, "<- Content-length: %ud\n", c->npostbody);
        }
    }
    if(c->authenticate){
        ioprint(io, fd, "Authorization: %s\r\n", c->authenticate);
        if(httpdebug)
            fprint(2, "<- Authorization: %s\n", c->authenticate);
    }
    ioprint(io, fd, "\r\n");
    if(c->havepostbody)
        if(iowrite(io, fd, c->postbody, c->npostbody) != c->npostbody)
            goto Error;

    redirect = 0;
    authenticate = 0;
    initibuf(&hs->b, io, fd);
    code = httprcode(hs);

    switch(code){
    case -1:    /* connection timed out */
        goto Error;

/*
    case Eof:
        werrstr("EOF from HTTP server");
        goto Error;
*/

    case 200:   /* OK */
    case 201:   /* Created */
    case 202:   /* Accepted */
    case 204:   /* No Content */
    case 205: /* Reset Content */
#ifdef NOT_DEFINED
        if(ofile == nil && r->start != 0)
            sysfatal("page changed underfoot");
#endif
        break;

    case 206:   /* Partial Content */
        werrstr("Partial Content (206)");
        goto Error;

    case 301:   /* Moved Permanently */
```

```
case 302:   /* Moved Temporarily */
case 303:   /* See Other */
case 307: /* Temporary Redirect  */
    redirect = 1;
    break;

case 304:   /* Not Modified */
    break;

case 400:   /* Bad Request */
    werrstr("Bad Request (400)");
    goto Error;

case 401:   /* Unauthorized */
    if(c->authenticate){
        werrstr("Authentication failed (401)");
        goto Error;
    }
    authenticate = 1;
    break;
case 402:   /* Payment Required */
    werrstr("Payment Required (402)");
    goto Error;

case 403:   /* Forbidden */
    werrstr("Forbidden by server (403)");
    goto Error;

case 404:   /* Not Found */
    werrstr("Not found on server (404)");
    goto Error;

case 405:   /* Method Not Allowed  */
    werrstr("Method not allowed (405)");
    goto Error;

case 406: /* Not Acceptable */
    werrstr("Not Acceptable (406)");
    goto Error;

case 407:   /* Proxy auth */
    werrstr("Proxy authentication required (407)");
    goto Error;

case 408: /* Request Timeout */
    werrstr("Request Timeout (408)");
    goto Error;

case 409: /* Conflict */
    werrstr("Conflict  (409)");
    goto Error;

case 410: /* Gone */
    werrstr("Gone  (410)");
    goto Error;

case 411: /* Length Required */
    werrstr("Length Required  (411)");
    goto Error;
```

```
case 412: /* Precondition Failed */
    werrstr("Precondition Failed  (412)");
    goto Error;

case 413: /* Request Entity Too Large */
    werrstr("Request Entity Too Large  (413)");
    goto Error;

case 414: /* Request-URI Too Long */
    werrstr("Request-URI Too Long  (414)");
    goto Error;

case 415: /* Unsupported Media Type */
    werrstr("Unsupported Media Type  (415)");
    goto Error;

case 416: /* Requested Range Not Satisfiable */
    werrstr("Requested Range Not Satisfiable  (416)");
    goto Error;

case 417: /* Expectation Failed */
    werrstr("Expectation Failed  (417)");
    goto Error;

case 500:   /* Internal server error */
    werrstr("Server choked (500)");
    goto Error;

case 501:   /* Not implemented */
    werrstr("Server can't do it (501)");
    goto Error;

case 502:   /* Bad gateway */
    werrstr("Bad gateway (502)");
    goto Error;

case 503:   /* Service unavailable */
    werrstr("Service unavailable (503)");
    goto Error;

default:
    /* Bogus: we should treat unknown code XYZ as code X00 */
    werrstr("Unknown response code %d", code);
    goto Error;
}

if(httpheaders(hs) < 0)
    goto Error;
if(c->ctl.acceptcookies && hs->setcookie)
    httpsetcookie(hs->setcookie, url->host, url->path);
if(authenticate){
    if(!hs->credentials){
        if(hs->autherror[0])
            werrstr("%s", hs->autherror);
        else
            werrstr("unauthorized; no www-authenticate: header");
        goto Error;
    }
    c->authenticate = hs->credentials;
    hs->credentials = nil;
```

```
    }else if(c->authenticate)
        c->authenticate = 0;
    if(redirect){
        if(!hs->location){
            werrstr("redirection without Location: header");
            goto Error;
        }
        c->redirect = hs->location;
        hs->location = nil;
    }
    return 0;
}
```

⟨*function* `httpread` *(webfs)* 102a⟩≡ (102c)

```
int
httpread(Client *c, Req *r)
{
    HttpState *hs;
    long n;

    hs = c->aux;
    n = readibuf(&hs->b, r->ofcall.data, r->ifcall.count);
    if(n < 0)
        return -1;

    r->ofcall.count = n;
    return 0;
}
```

⟨*function* `httpclose` *(webfs)* 102b⟩≡ (102c)

```
void
httpclose(Client *c)
{
    HttpState *hs;

    hs = c->aux;
    if(hs == nil)
        return;
    if(hs->fd >= 0)
        ioclose(c->io, hs->fd);
    hs->fd = -1;
    free(hs->location);
    free(hs->setcookie);
    free(hs->netaddr);
    free(hs->credentials);
    free(hs);
    c->aux = nil;
}
```

⟨`webfs/http.c` 102c⟩≡

```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ip.h>
#include <plumb.h>
#include <thread.h>
#include <fcall.h>
#include <9p.h>
#include <libsec.h>
#include <auth.h>
```

```
#include "dat.h"
#include "fns.h"

char PostContentType[] = "application/x-www-form-urlencoded";
int httpdebug;

typedef struct HttpState HttpState;
```

⟨*struct* `HttpState` *(webfs)* 93⟩

⟨*function* `location` *(webfs)* 94b⟩

⟨*function* `contenttype` *(webfs)* 94c⟩

⟨*function* `setcookie` *(webfs)* 94d⟩

⟨*function* `unquote` *(webfs)* 94e⟩

⟨*function* `servername` *(webfs)* 95a⟩

⟨*function* `wwwauthenticate` *(webfs)* 95b⟩

⟨*global* `hdrtab` *(webfs)* 94a⟩

⟨*function* `httprcode` *(webfs)* 96⟩

⟨*function* `getheader` *(webfs)* 97a⟩

⟨*function* `httpheaders` *(webfs)* 97b⟩

⟨*function* `httpopen` *(webfs)* 98⟩

⟨*function* `httpread` *(webfs)* 102a⟩

⟨*function* `httpclose` *(webfs)* 102b⟩

webfs/io.c

⟨*function* `_iovfprint` *(webfs)* 103a⟩≡ (105a)

```
static long
_iovfprint(va_list *arg)
{
    int fd;
    char *fmt;
    va_list arg2;

    fd = va_arg(*arg, int);
    fmt = va_arg(*arg, char*);
    arg2 = va_arg(*arg, va_list);
    return vfprint(fd, fmt, arg2);
}
```

⟨*function* `iovfprint` *(webfs)* 103b⟩≡ (105a)

```
int
iovfprint(Ioproc *io, int fd, char *fmt, va_list arg)
{
    return iocall(io, _iovfprint, fd, fmt, arg);
}
```

⟨*function* `ioprint` *(webfs)* 104a⟩≡ (105a)

```
int
ioprint(Ioproc *io, int fd, char *fmt, ...)
{
    int n;
    va_list arg;

    va_start(arg, fmt);
    n = iovfprint(io, fd, fmt, arg);
    va_end(arg);
    return n;
}
```

⟨*function* `_iotlsdial` *(webfs)* 104b⟩≡ (105a)

```
static long
_iotlsdial(va_list *arg)
{
    char *addr, *local, *dir, *servername;
    int *cfdp, fd, tfd, usetls;
    TLSconn conn;

    addr = va_arg(*arg, char*);
    local = va_arg(*arg, char*);
    dir = va_arg(*arg, char*);
    cfdp = va_arg(*arg, int*);
    usetls = va_arg(*arg, int);
    servername = va_arg(*arg, char*);

    fd = dial(addr, local, dir, cfdp);
    if(fd < 0)
        return -1;
    if(!usetls)
        return fd;

    memset(&conn, 0, sizeof conn);
    /* does no good, so far anyway */
    // conn.chain = readcertchain("/sys/lib/ssl/vsignss.pem");
    /* claude: SNI is required by most modern HTTPS servers */
    if(servername != nil)
        conn.serverName = strdup(servername);

    tfd = tlsClient(fd, &conn);
    close(fd);
    if(tfd < 0)
        fprint(2, "%s: tlsClient: %r\n", argv0);
    else {
        /* BUG: check cert here? */
        if(conn.cert)
            free(conn.cert);
    }
    free(conn.serverName);
    return tfd;
}
```

⟨*function* `iotlsdial` *(webfs)* 104c⟩≡ (105a)

```
int
iotlsdial(Ioproc *io, char *addr, char *local, char *dir, int *cfdp, int usetls, char *servername)
{
    return iocall(io, _iotlsdial, addr, local, dir, cfdp, usetls, servername);
}
```

⟨webfs/io.c 105a⟩≡
```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ip.h>
#include <plumb.h>
#include <thread.h>
#include <fcall.h>
#include <9p.h>
#include <mp.h>
#include <libsec.h>
#include "dat.h"
#include "fns.h"
```

⟨*function* _iovfprint*(webfs)* 103a⟩

⟨*function* iovfprint*(webfs)* 103b⟩

⟨*function* ioprint*(webfs)* 104a⟩

⟨*function* _iotlsdial*(webfs)* 104b⟩

⟨*function* iotlsdial*(webfs)* 104c⟩

webfs/main.c

⟨*global* globalctl*(webfs)* 105b⟩≡ (106a)
```
Ctl globalctl =
{
    1,  /* accept cookies */
    1,  /* send cookies */
    10, /* redirect limit */
    "webfs/2.0 (plan 9)"        /* user agent */
};
```

⟨*function* usage*(webfs)* 105c⟩≡ (106a)
```
void
usage(void)
{
    fprint(2, "usage: webfs [-c cookies] [-m mtpt] [-s service]\n");
    threadexitsall("usage");
}
```

⟨*function* threadmain*(webfs)* 105d⟩≡ (106a)
```
void
threadmain(int argc, char **argv)
{
    rfork(RFNOTEG);
    ARGBEGIN{
    case 'd':
        mainmem->flags |= POOL_PARANOIA|POOL_ANTAGONISM;
        break;
    case 'D':
        chatty9p++;
        break;
    case 'c':
        cookiefile = EARGF(usage());
        break;
    case 'm':
```

```
            mtpt = EARGF(usage());
            break;
        case 's':
            service = EARGF(usage());
            break;
        default:
            usage();
        }ARGEND


        quotefmtinstall();
        if(argc != 0)
            usage();

        plumbinit();
        globalctl.useragent = estrdup(globalctl.useragent);
        initcookies(cookiefile);
        initurl();
        initfs();
        threadpostmountsrv(&fs, service, mtpt, MREPL);
        threadexits(nil);
    }
```

⟨webfs/main.c 106a⟩≡

```
    #include <u.h>
    #include <libc.h>
    #include <bio.h>
    #include <ip.h>
    #include <plumb.h>
    #include <thread.h>
    #include <fcall.h>
    #include <9p.h>
    #include "dat.h"
    #include "fns.h"

    char *cookiefile;
    char *mtpt = "/mnt/web";
    char *service;

    ⟨global globalctl(webfs) 105b⟩

    ⟨function usage(webfs) 105c⟩

    #include <pool.h>
    ⟨function threadmain(webfs) 105d⟩
```

webfs/plumb.c

⟨global ctypes(webfs) 106b⟩≡ (109b)

```
    static struct
    {
        char        *ctype;
        char        *ext;
    }
    ctypes[] =
    {
        { "application/msword", "doc" },
        { "application/pdf", "pdf" },
        { "application/postscript", "ps" },
```

```
    { "application/rtf", "rtf" },
    { "image/gif", "gif" },
    { "image/jpeg", "jpg" },
    { "image/png", "png" },
    { "image/ppm", "ppm" },
    { "image/tiff", "tiff" },
    { "text/html", "html" },
    { "text/plain", "txt" },
    { "text/xml", "xml" },
};
```

⟨*function* `plumbinit` *(webfs)* 107a⟩≡ (109b)

```
void
plumbinit(void)
{
    plumbsendfd = plumbopen("send", OWRITE|OCEXEC);
    plumbwebfd = plumbopen("web", OREAD|OCEXEC);
}
```

⟨*function* `plumbstart` *(webfs)* 107b⟩≡ (109b)

```
void
plumbstart(void)
{
    plumbchan = chancreate(sizeof(Plumbmsg*), 0);
    proccreate(plumbwebproc, nil, STACK);
    threadcreate(plumbwebthread, nil, STACK);
}
```

⟨*function* `plumbwebthread` *(webfs)* 107c⟩≡ (109b)

```
static void
plumbwebthread(void*)
{
    char *base;
    Plumbmsg *m;

    for(;;){
        m = recvp(plumbchan);
        if(m == nil)
            threadexits(nil);
        base = plumblookup(m->attr, "baseurl");
        if(base == nil)
            base = m->wdir;
        plumburl(m->data, base);
        plumbfree(m);
    }
}
```

⟨*function* `plumbwebproc` *(webfs)* 107d⟩≡ (109b)

```
static void
plumbwebproc(void*)
{
    Plumbmsg *m;

    for(;;){
        m = plumbrecv(plumbwebfd);
        sendp(plumbchan, m);
        if(m == nil)
            threadexits(nil);
    }
}
```

⟨*function* `addattr` *(webfs)* 108a⟩≡ (109b)

```
static void
addattr(Plumbmsg *m, char *name, char *value)
{
    Plumbattr *a;

    a = malloc(sizeof(Plumbattr));
    a->name = name;
    a->value = value;
    a->next = m->attr;
    m->attr = a;
}
```

⟨*function* `freeattrs` *(webfs)* 108b⟩≡ (109b)

```
static void
freeattrs(Plumbmsg *m)
{
    Plumbattr *a, *next;

    a = m->attr;
    while(a != nil) {
        next = a->next;
        free(a);
        a = next;
    }
}
```

⟨*function* `replumb` *(webfs)* 108c⟩≡ (109b)

```
void
replumb(Client *c)
{
    int i;
    Plumbmsg *m;
    char name[128], *ctype, *ext, *p;

    if(!c->plumbed)
        return;
    m = emalloc(sizeof(Plumbmsg));
    m->src = "webfs";
    m->dst = nil;
    m->wdir = "/";
    m->type = "text";
    m->attr = nil;
    addattr(m, "url", c->url->url);
    ctype = c->contenttype;
    ext = nil;
    if(ctype != nil) {
        addattr(m, "content-type", ctype);
        for(i = 0; i < nelem(ctypes); i++) {
            if(strcmp(ctype, ctypes[i].ctype) == 0) {
                ext = ctypes[i].ext;
                break;
            }
        }
    }
    if(ext == nil) {
        p = strrchr(c->url->url, '/');
        if(p != nil)
            p = strrchr(p+1, '.');
        if(p != nil && strlen(p) <= 5)
```

```
            ext = p+1;
        else
            ext = "txt";                /* punt */
    }
    c->ext = ext;
  if(0)fprint(2, "content type %s -> extension .%s\n", ctype, ext);
    m->ndata = snprint(name, sizeof name, "/mnt/web/%d/body.%s", c->num, ext);
    m->data = estrdup(name);
    proccreate(plumbsendproc, m, STACK);        /* separate proc to avoid a deadlock */
  }
```

⟨*function* `plumbsendproc` *(webfs)* 109a⟩≡ (109b)

```
  static void
  plumbsendproc(void *x)
  {
      Plumbmsg *m;

      m = x;
      plumbsend(plumbsendfd, m);
      freeattrs(m);
      free(m->data);
      free(m);
  }
```

⟨`webfs/plumb.c` 109b⟩≡

```
  #include <u.h>
  #include <libc.h>
  #include <auth.h>
  #include <fcall.h>
  #include <thread.h>
  #include <plumb.h>
  #include <9p.h>

  #include "dat.h"
  #include "fns.h"

  static int              plumbsendfd;
  static int              plumbwebfd;
  static Channel  *plumbchan;

  static void     plumbwebproc(void*);
  static void     plumbwebthread(void*);
  static void plumbsendproc(void*);
```

⟨*function* `plumbinit` *(webfs)* 107a⟩

⟨*function* `plumbstart` *(webfs)* 107b⟩

⟨*function* `plumbwebthread` *(webfs)* 107c⟩

⟨*function* `plumbwebproc` *(webfs)* 107d⟩

⟨*function* `addattr` *(webfs)* 108a⟩

⟨*function* `freeattrs` *(webfs)* 108b⟩

⟨*global* `ctypes` *(webfs)* 106b⟩

⟨*function* `replumb` *(webfs)* 108c⟩

⟨*function* `plumbsendproc` *(webfs)* 109a⟩

webfs/url.c

⟨*global* schemestrtab*(webfs)* 110a⟩≡ (127b)

```
static char*
schemestrtab[] =
{
    nil,
    "http",
    "https",
    "ftp",
    "file",
};
```

⟨*struct* Retab*(webfs)* 110b⟩≡ (127b)

```
struct Retab
{
    char        *str;
    Reprog      *prog;
    int         size;
    int         ind[5];
};
```

⟨*enum* RExxx*(webfs)* 110c⟩≡ (127b)

```
enum
{
    REsplit = 0,
    REscheme,
    REunknowndata,
    REauthority,
    REhost,
    REuserinfo,
    REabspath,
    REquery,
    REfragment,
    REhttppath,
    REftppath,
    REfilepath,

    MaxResub=   20,
};
```

⟨*global* retab*(webfs)* 110d⟩≡ (127b)

```
Retab retab[] = /* view in constant width Font */
{
[REsplit]
    "^(([^:/?#]+):)?(//([^/?#]*))?([^?#]+)?(\\?([^#]*))?(#(.*))?$", nil, 0,
    /* |-scheme-|      |-auth.-|  |path--|    |query|     |--|frag */
    {  2,              4,         5,          7,          9},

[REscheme]
    "^[a-z][a-z0-9+-.]*$", nil, 0,
    { 0, },

[REunknowndata]
    "^" URICNOSLASH_2 URIC_2 "*$", nil, 0,
    { 0, },

[REauthority]
    "^((" USERINFO_2 "*)@)?(((\\[[^\\]@]+\\])|([^:\\[@]+))(:([0-9]*))?)?)?$", nil, 0,
    /* |----user info-----|  |--------host----------------|  |-port-| */
```

```
    {  3,                    7,                                11, },

  [REhost]
    "^(([a-zA-Z0-9\\-.]+)|(\\[([a-fA-F0-9.:]+)\\]))$", nil, 0,
    /* |--regular host--|     |-IPv6 literal-| */
    {  2,                     4, },

  [REuserinfo]
    "^(([^:]*)(:([^:]*))?)$", nil, 0,
    /* |user-|  |pass-| */
    {  2,       4, },

  [REabspath]
    "^/" PSEGCHAR_3 "*$", nil, 0,
    { 0, },

  [REquery]
    "^" URIC_2 "*$", nil, 0,
    { 0, },

  [REfragment]
    "^" URIC_2 "*$", nil, 0,
    { 0, },

  [REhttppath]
    "^.*$", nil, 0,
    { 0, },

  [REftppath]
    "^(.+)(;[tT][yY][pP][eE]=([aAiIdD]))?$", nil, 0,
    /*|--|-path              |ftptype-| */
    { 1,                     3, },

  [REfilepath]
    "^.*$", nil, 0,
    { 0, },
  };
```

⟨*struct* SplitUrl*(webfs)* 111a⟩≡ (127b)

```
  struct SplitUrl
  {
      struct {
          char *s;
          char *e;
      } url, scheme, authority, path, query, fragment;
  };
```

⟨*constant* RemoveExtraRelDotDots*(webfs)* 111b⟩≡ (127b)

```
  #define RemoveExtraRelDotDots   0
```

⟨*constant* ExpandCurrentDocUrls*(webfs)* 111c⟩≡ (127b)

```
  #define ExpandCurrentDocUrls    1
```

⟨*function* ischeme*(webfs)* 111d⟩≡ (127b)

```
  static int
  ischeme(char *s)
  {
      int i;

      for(i=0; i<nelem(schemestrtab); i++)
```

```
        if(schemestrtab[i] && strcmp(s, schemestrtab[i])==0)
            return i;
    return USunknown;
}
```

⟨*constant* PUNCT*(webfs)* 112a⟩≡ (127b)
```
#define PUNCT                   "\\-_.!~*'()"
```

⟨*constant* RES*(webfs)* 112b⟩≡ (127b)
```
#define RES                     ";/?:@&=+$,"
```

⟨*constant* ALNUM*(webfs)* 112c⟩≡ (127b)
```
#define ALNUM           "a-zA-Z0-9"
```

⟨*constant* HEX*(webfs)* 112d⟩≡ (127b)
```
#define HEX                     "0-9a-fA-F"
```

⟨*constant* UNRES*(webfs)* 112e⟩≡ (127b)
```
#define UNRES                   ALNUM PUNCT
```

⟨*constant* ESCAPED_1*(webfs)* 112f⟩≡ (127b)
```
#define ESCAPED_1                       "(%[" HEX "][" HEX "])"
```

⟨*constant* URIC_2*(webfs)* 112g⟩≡ (127b)
```
#define URIC_2                  "([" RES UNRES "]|" ESCAPED_1 ")"
```

⟨*constant* URICNOSLASH_2*(webfs)* 112h⟩≡ (127b)
```
#define URICNOSLASH_2           "([" UNRES ";?:@&=+$,]|" ESCAPED_1 ")"
```

⟨*constant* USERINFO_2*(webfs)* 112i⟩≡ (127b)
```
#define USERINFO_2              "([" UNRES ";:&=+$,]|" ESCAPED_1 ")"
```

⟨*constant* PCHAR_2*(webfs)* 112j⟩≡ (127b)
```
#define PCHAR_2                 "([" UNRES ":@&=+$,]|" ESCAPED_1 ")"
```

⟨*constant* PSEGCHAR_3*(webfs)* 112k⟩≡ (127b)
```
#define PSEGCHAR_3              "([/;]|" PCHAR_2 ")"
```

⟨*function* countleftparen*(webfs)* 112l⟩≡ (127b)
```
static int
countleftparen(char *s)
{
    int n;

    n = 0;
    for(; *s; s++)
        if(*s == '(')
            n++;
    return n;
}
```

⟨*function* `initurl` *(webfs)* 113a⟩≡ (127b)

```
void
initurl(void)
{
    int i, j;

    for(i=0; i<nelem(retab); i++){
        retab[i].prog = regcomp(retab[i].str);
        if(retab[i].prog == nil)
            sysfatal("recomp(%s): %r", retab[i].str);
        retab[i].size = countleftparen(retab[i].str)+1;
        for(j=0; j<nelem(retab[i].ind); j++)
            if(retab[i].ind[j] >= retab[i].size)
                sysfatal("bad index in regexp table: retab[%d].ind[%d] = %d >= %d",
                    i, j, retab[i].ind[j], retab[i].size);
        if(MaxResub < retab[i].size)
            sysfatal("MaxResub too small: %d < %d", MaxResub, retab[i].size);
    }
}
```

⟨*function* `merge_relative_path` *(webfs)* 113b⟩≡ (127b)

```
/*
 * Implements the algorithm in RFC2396 sec 5.2 step 6.
 * Returns number of chars written, excluding NUL terminator.
 * dest is known to be >= strlen(base)+rel_len.
 */
static void
merge_relative_path(char *base, char *rel_st, int rel_len, char *dest)
{
    char *s, *p, *e, *pdest;

    pdest = dest;

    /* 6a: start with base, discard last segment */
    if(base && base[0]){
        /* Empty paths don't match in our scheme; 'base' should be nil */
        assert(base[0] == '/');
        e = strrchr(base, '/');
        e++;
        memmove(pdest, base, e-base);
        pdest += e-base;
    }else{
        /* Artistic license on my part */
        *pdest++ = '/';
    }

    /* 6b: append relative component */
    if(rel_st){
        memmove(pdest, rel_st, rel_len);
        pdest += rel_len;
    }

    /* 6c: remove any occurrences of "./" as a complete segment */
    s = dest;
    *pdest = '\0';
    while(e = strstr(s, "./")){
        if((e == dest) || (*(e-1) == '/')){
                        memmove(e, e+2, pdest+1-(e+2)); /* +1 for NUL */
            pdest -= 2;
        }else
```

```
            s = e+1;
    }

    /* 6d: remove a trailing "." as a complete segment */
    if(pdest>dest && *(pdest-1)=='.' &&
      (pdest==dest+1 || *(pdest-2)=='/'))
        *--pdest = '\0';

    /* 6e: remove occurences of "seg/../", where seg != "..", left->right */
    s = dest+1;
    while(e = strstr(s, "/../")){
        p = e - 1;
        while(p >= dest && *p != '/')
            p--;
        if(memcmp(p, "/../", 4) != 0){
            memmove(p+1, e+4, pdest+1-(e+4));
            pdest -= (e+4) - (p+1);
        }else
            s = e+1;
    }

    /* 6f: remove a trailing "seg/..", where seg isn't ".."  */
    if(pdest-3 > dest && memcmp(pdest-3, "/..", 3)==0){
        p = pdest-3 - 1;
        while(p >= dest && *p != '/')
            p--;
        if(memcmp(p, "/../", 4) != 0){
            pdest = p+1;
            *pdest = '\0';
        }
    }

    /* 6g: leading ".." segments are errors -- we'll just blat them out. */
    if(RemoveExtraRelDotDots){
        p = dest;
        if (p[0] == '/')
            p++;
        s = p;
        while(s[0]=='.' && s[1]=='.' && (s[2]==0 || s[2]=='/'))
            s += 3;
        if(s > p){
            memmove(p, s, pdest+1-s);
            pdest -= s-p;
        }
    }
    USED(pdest);

    if(urldebug)
        fprint(2, "merge_relative_path: '%s' + '%.*s' -> '%s'\n", base, rel_len,
            rel_st, dest);
}
```

⟨*function* `resolve_relative` *(webfs)* 114⟩≡ (127b)

```
/*
 * See RFC2396 sec 5.2 for info on resolving relative URIs to absolute form.
 *
 * If successful, this just ends up freeing and replacing "u->url".
 */
static int
resolve_relative(SplitUrl *su, Url *base, Url *u)
```

```
{
    char *url, *path;
    char *purl, *ppath;
    int currentdoc, ulen, plen;

    if(base == nil){
        werrstr("relative URI given without base");
        return -1;
    }
    if(base->scheme == nil){
        werrstr("relative URI given with no scheme");
        return -1;
    }
    if(base->ischeme == USunknown){
        werrstr("relative URI given with unknown scheme");
        return -1;
    }
    if(base->ischeme == UScurrent){
        werrstr("relative URI given with incomplete base");
        return -1;
    }
    assert(su->scheme.s == nil);

    /* Sec 5.2 step 2 */
    currentdoc = 0;
    if(su->path.s==nil && su->scheme.s==nil && su->authority.s==nil && su->query.s==nil){
        /* Reference is to current document */
        if(urldebug)
            fprint(2, "url %s is relative to current document\n", u->url);
        u->ischeme = UScurrent;
        if(!ExpandCurrentDocUrls)
            return 0;
        currentdoc = 1;
    }

    /* Over-estimate the maximum lengths, for allocation purposes */
    /* (constants are for separators) */
    plen = 1;
    if(base->path)
        plen += strlen(base->path);
    if(su->path.s)
        plen += 1 + (su->path.e - su->path.s);

    ulen = 0;
    ulen += strlen(base->scheme) + 1;
    if(su->authority.s)
        ulen += 2 + (su->authority.e - su->authority.s);
    else
        ulen += 2 + ((base->authority) ? strlen(base->authority) : 0);
    ulen += plen;
    if(su->query.s)
        ulen += 1 + (su->query.e - su->query.s);
    else if(currentdoc && base->query)
        ulen += 1 + strlen(base->query);
    if(su->fragment.s)
        ulen += 1 + (su->fragment.e - su->fragment.s);
    else if(currentdoc && base->fragment)
        ulen += 1 + strlen(base->fragment);
    url = emalloc(ulen+1);
    path = emalloc(plen+1);
```

```
url[0] = '\0';
purl = url;
path[0] = '\0';
ppath = path;

if(su->authority.s || (su->path.s && (su->path.s[0] == '/'))){
    /* Is a "network-path" or "absolute-path"; don't merge with base path */
    /* Sec 5.2 steps 4,5 */
    if(su->path.s){
        memmove(ppath, su->path.s, su->path.e - su->path.s);
        ppath += su->path.e - su->path.s;
        *ppath = '\0';
    }
}else if(currentdoc){
    /* Is a current-doc reference; just copy the path from the base URL */
    if(base->path){
        strcpy(ppath, base->path);
        ppath += strlen(ppath);
    }
    USED(ppath);
}else{
    /* Is a relative-path reference; we have to merge it */
    /* Sec 5.2 step 6 */
    merge_relative_path(base->path,
        su->path.s, su->path.e - su->path.s, ppath);
}

/* Build new URL from pieces, inheriting from base where needed */
strcpy(purl, base->scheme);
purl += strlen(purl);
*purl++ = ':';
if(su->authority.s){
    strcpy(purl, "//");
    purl += strlen(purl);
    memmove(purl, su->authority.s, su->authority.e - su->authority.s);
    purl += su->authority.e - su->authority.s;
}else if(base->authority){
    strcpy(purl, "//");
    purl += strlen(purl);
    strcpy(purl, base->authority);
    purl += strlen(purl);
}
assert((path[0] == '\0') || (path[0] == '/'));
strcpy(purl, path);
purl += strlen(purl);

/*
 * The query and fragment are not inherited from the base,
 * except in case of "current document" URLs, which inherit any query
 * and may inherit the fragment.
 */
if(su->query.s){
    *purl++ = '?';
    memmove(purl, su->query.s, su->query.e - su->query.s);
    purl += su->query.e - su->query.s;
}else if(currentdoc && base->query){
    *purl++ = '?';
    strcpy(purl, base->query);
    purl += strlen(purl);
```

```
    }

    if(su->fragment.s){
        *purl++ = '#';
        memmove(purl, su->query.s, su->query.e - su->query.s);
        purl += su->fragment.e - su->fragment.s;
    }else if(currentdoc && base->fragment){
        *purl++ = '#';
        strcpy(purl, base->fragment);
        purl += strlen(purl);
    }
    USED(purl);

    if(urldebug)
        fprint(2, "resolve_relative: '%s' + '%s' -> '%s'\n", base->url, u->url, url);
    free(u->url);
    u->url = url;
    free(path);
    return 0;
}
```

⟨*function* `regx` *(webfs)* 117a⟩≡ (127b)

```
int
regx(Reprog *prog, char *s, Resub *m, int nm)
{
    int i;

    if(s == nil)
        s = m[0].s.sp;  /* why is this necessary? */

    i = regexec(prog, s, m, nm);
/*
    if(i >= 0)
        for(j=0; j<nm; j++)
            fprint(2, "match%d: %.*s\n", j, utfnlen(m[j].s.sp, m[j].e.ep-m[j].s.sp), m[j].s.sp);
*/
    return i;
}
```

⟨*function* `ismatch` *(webfs)* 117b⟩≡ (127b)

```
static int
ismatch(int i, char *s, char *desc)
{
    Resub m[1];

    m[0].s.sp = m[0].e.ep = nil;
    if(!regx(retab[i].prog, s, m, 1)){
        werrstr("malformed %s: %q", desc, s);
        return 0;
    }
    return 1;
}
```

⟨*function* `spliturl` *(webfs)* 117c⟩≡ (127b)

```
static int
spliturl(char *url, SplitUrl *su)
{
    Resub m[MaxResub];
    Retab *t;
```

```
    /*
     * Newlines are not valid in a URI, but regexp(2) treats them specially
     * so it's best to make sure there are none before proceeding.
     */
    if(strchr(url, '\n')){
        werrstr("newline in URI");
        return -1;
    }

    /*
     * Because we use NUL-terminated strings, as do many client and server
     * implementations, an escaped NUL ("%00") will quite likely cause problems
     * when unescaped.  We can check for such a sequence once before examining
         * the components because, per RFC2396 sec. 2.4.1 - 2.4.2, '%' is reserved
     * in URIs to _always_ indicate escape sequences.  Something like "%2500"
     * will still get by, but that's legitimate, and if it ends up causing
     * a NUL then someone is unescaping too many times.
     */
    if(strstr(url, "%00")){
        werrstr("escaped NUL in URI");
        return -1;
    }

    m[0].s.sp = m[0].e.ep = nil;
    t = &retab[REsplit];
    if(!regx(t->prog, url, m, t->size)){
        werrstr("malformed URI: %q", url);
        return -1;
    }

    su->url.s = m[0].s.sp;
    su->url.e = m[0].e.ep;
    su->scheme.s = m[t->ind[0]].s.sp;
    su->scheme.e = m[t->ind[0]].e.ep;
    su->authority.s = m[t->ind[1]].s.sp;
    su->authority.e = m[t->ind[1]].e.ep;
    su->path.s = m[t->ind[2]].s.sp;
    su->path.e = m[t->ind[2]].e.ep;
    su->query.s = m[t->ind[3]].s.sp;
    su->query.e = m[t->ind[3]].e.ep;
    su->fragment.s = m[t->ind[4]].s.sp;
    su->fragment.e = m[t->ind[4]].e.ep;

    if(urldebug)
        fprint(2, "split url %s into %.*q %.*q %.*q %.*q %.*q\n",
            url,
            su->url.s ? utfnlen(su->url.s, su->url.e-su->url.s) : 10, su->url.s ? su->url.s : "",
            su->scheme.s ? utfnlen(su->scheme.s, su->scheme.e-su->scheme.s) : 10, su->scheme.s ? su->scheme.s :
            su->authority.s ? utfnlen(su->authority.s, su->authority.e-su->authority.s) : 10, su->authority.s ?
            su->path.s ? utfnlen(su->path.s, su->path.e-su->path.s) : 10, su->path.s ? su->path.s : "",
            su->query.s ? utfnlen(su->query.s, su->query.e-su->query.s) : 10, su->query.s ? su->query.s : "",
            su->fragment.s ? utfnlen(su->fragment.s, su->fragment.e-su->fragment.s) : 10, su->fragment.s ? su->

    return 0;
}
```

⟨*function* parse_scheme*(webfs)* 118⟩≡ (127b)

```
static int
parse_scheme(SplitUrl *su, Url *u)
{
```

```
    if(su->scheme.s == nil){
        werrstr("missing scheme");
        return -1;
    }
    u->scheme = estredup(su->scheme.s, su->scheme.e);
    strlower(u->scheme);

    if(!ismatch(REscheme, u->scheme, "scheme"))
        return -1;

    u->ischeme = ischeme(u->scheme);
    if(urldebug)
        fprint(2, "parse_scheme %s => %d\n", u->scheme, u->ischeme);
    return 0;
}
```

⟨*function* parse_unknown_part*(webfs)* 119a⟩≡ (127b)

```
static int
parse_unknown_part(SplitUrl *su, Url *u)
{
    char *s, *e;

    assert(u->ischeme == USunknown);
    assert(su->scheme.e[0] == ':');

    s = su->scheme.e+1;
    if(su->fragment.s){
        e = su->fragment.s-1;
        assert(*e == '#');
    }else
        e = s+strlen(s);

    u->schemedata = estredup(s, e);
    if(!ismatch(REunknowndata, u->schemedata, "unknown scheme data"))
        return -1;
    return 0;
}
```

⟨*function* parse_userinfo*(webfs)* 119b⟩≡ (127b)

```
static int
parse_userinfo(char *s, char *e, Url *u)
{
    Resub m[MaxResub];
    Retab *t;

    m[0].s.sp = s;
    m[0].e.ep = e;
    t = &retab[REuserinfo];
    if(!regx(t->prog, nil, m, t->size)){
        werrstr("malformed userinfo: %.*q", utfnlen(s, e-s), s);
        return -1;
    }
    if(m[t->ind[0]].s.sp)
        u->user = estredup(m[t->ind[0]].s.sp, m[t->ind[0]].e.ep);
    if(m[t->ind[1]].s.sp)
        u->user = estredup(m[t->ind[1]].s.sp, m[t->ind[1]].e.ep);
    return 0;
}
```

⟨*function* parse_host*(webfs)* 120a⟩≡ (127b)

```
static int
parse_host(char *s, char *e, Url *u)
{
    Resub m[MaxResub];
    Retab *t;

    m[0].s.sp = s;
    m[0].e.ep = e;
    t = &retab[REhost];
    if(!regx(t->prog, nil, m, t->size)){
        werrstr("malformed host: %.*q", utfnlen(s, e-s), s);
        return -1;
    }

    assert(m[t->ind[0]].s.sp || m[t->ind[1]].s.sp);

    if(m[t->ind[0]].s.sp)       /* regular */
        u->host = estredup(m[t->ind[0]].s.sp, m[t->ind[0]].e.ep);
    else
        u->host = estredup(m[t->ind[1]].s.sp, m[t->ind[1]].e.ep);
    return 0;
}
```

⟨*function* parse_authority*(webfs)* 120b⟩≡ (127b)

```
static int
parse_authority(SplitUrl *su, Url *u)
{
    Resub m[MaxResub];
    Retab *t;
    char *host;
    char *userinfo;

    if(su->authority.s == nil)
        return 0;

    u->authority = estredup(su->authority.s, su->authority.e);
    m[0].s.sp = m[0].e.ep = nil;
    t = &retab[REauthority];
    if(!regx(t->prog, u->authority, m, t->size)){
        werrstr("malformed authority: %q", u->authority);
        return -1;
    }

    if(m[t->ind[0]].s.sp)
        if(parse_userinfo(m[t->ind[0]].s.sp, m[t->ind[0]].e.ep, u) < 0)
            return -1;
    if(m[t->ind[1]].s.sp)
        if(parse_host(m[t->ind[1]].s.sp, m[t->ind[1]].e.ep, u) < 0)
            return -1;
    if(m[t->ind[2]].s.sp)
        u->port = estredup(m[t->ind[2]].s.sp, m[t->ind[2]].e.ep);


    if(urldebug > 0){
        userinfo = estredup(m[t->ind[0]].s.sp, m[t->ind[0]].e.ep);
        host = estredup(m[t->ind[1]].s.sp, m[t->ind[1]].e.ep);
        fprint(2, "port: %q, authority %q\n", u->port, u->authority);
        fprint(2, "host %q, userinfo %q\n", host, userinfo);
        free(host);
```

```
        free(userinfo);
    }
    return 0;
}
```

⟨*function* parse_abspath*(webfs)* 121a⟩≡ (127b)

```
static int
parse_abspath(SplitUrl *su, Url *u)
{
    if(su->path.s == nil)
        return 0;
    u->path = estredup(su->path.s, su->path.e);
    if(!ismatch(REabspath, u->path, "absolute path"))
        return -1;
    return 0;
}
```

⟨*function* parse_query*(webfs)* 121b⟩≡ (127b)

```
static int
parse_query(SplitUrl *su, Url *u)
{
    if(su->query.s == nil)
        return 0;
    u->query = estredup(su->query.s, su->query.e);
    if(!ismatch(REquery, u->query, "query"))
        return -1;
    return 0;
}
```

⟨*function* parse_fragment*(webfs)* 121c⟩≡ (127b)

```
static int
parse_fragment(SplitUrl *su, Url *u)
{
    if(su->fragment.s == nil)
        return 0;
    u->fragment = estredup(su->fragment.s, su->fragment.e);
    if(!ismatch(REfragment, u->fragment, "fragment"))
        return -1;
    return 0;
}
```

⟨*function* postparse_http*(webfs)* 121d⟩≡ (127b)

```
static int
postparse_http(Url *u)
{
    u->open = httpopen;
    u->read = httpread;
    u->close = httpclose;

    if(u->authority==nil){
        werrstr("missing authority (hostname, port, etc.)");
        return -1;
    }
    if(u->host == nil){
        werrstr("missing host specification");
        return -1;
    }

    if(u->path == nil){
        u->http.page_spec = estrdup("/");
```

```
        return 0;
    }

    if(!ismatch(REhttppath, u->path, "http path"))
        return -1;
    if(u->query){
        u->http.page_spec = emalloc(strlen(u->path)+1+strlen(u->query)+1);
        strcpy(u->http.page_spec, u->path);
        strcat(u->http.page_spec, "?");
        strcat(u->http.page_spec, u->query);
    }else
        u->http.page_spec = estrdup(u->path);

    return 0;
}
```

⟨*function* postparse_ftp*(webfs)* 122⟩≡ (127b)

```
static int
postparse_ftp(Url *u)
{
    Resub m[MaxResub];
    Retab *t;

    if(u->authority==nil){
        werrstr("missing authority (hostname, port, etc.)");
        return -1;
    }
    if(u->query){
        werrstr("unexpected \"?query\" in ftp path");
        return -1;
    }
    if(u->host == nil){
        werrstr("missing host specification");
        return -1;
    }

    if(u->path == nil){
        u->ftp.path_spec = estrdup("/");
        return 0;
    }

    m[0].s.sp = m[0].e.ep = nil;
    t = &retab[REftppath];
    if(!regx(t->prog, u->path, m, t->size)){
        werrstr("malformed ftp path: %q", u->path);
        return -1;
    }

    if(m[t->ind[0]].s.sp){
        u->ftp.path_spec = estredup(m[t->ind[0]].s.sp, m[t->ind[0]].e.ep);
        if(strchr(u->ftp.path_spec, ';')){
            werrstr("unexpected \";param\" in ftp path");
            return -1;
        }
    }else
        u->ftp.path_spec = estrdup("/");

    if(m[t->ind[1]].s.sp){
        u->ftp.type = estredup(m[t->ind[1]].s.sp, m[t->ind[1]].e.ep);
        strlower(u->ftp.type);
```

```
    }
    return 0;
}
```

⟨*function* `postparse_file` *(webfs)* 123a⟩≡ (127b)

```
static int
postparse_file(Url *u)
{
    if(u->user || u->passwd){
        werrstr("user information not valid with file scheme");
        return -1;
    }
    if(u->query){
        werrstr("unexpected \"?query\" in file path");
        return -1;
    }
    if(u->port){
        werrstr("port not valid with file scheme");
        return -1;
    }
    if(u->path == nil){
        werrstr("missing path in file scheme");
        return -1;
    }
    if(strchr(u->path, ';')){
        werrstr("unexpected \";param\" in file path");
        return -1;
    }

    if(!ismatch(REfilepath, u->path, "file path"))
        return -1;

    /* "localhost" is equivalent to no host spec, we'll chose the latter */
    if(u->host && cistrcmp(u->host, "localhost") == 0){
        free(u->host);
        u->host = nil;
    }
    return 0;
}
```

⟨*function* `parseurl` *(webfs)* 123b⟩≡ (127b)

```
Url*
parseurl(char *url, Url *base)
{
    Url *u;
    SplitUrl su;

    if(urldebug)
        fprint(2, "parseurl %s with base %s\n", url, base ? base->url : "<none>");

    u = emalloc(sizeof(Url));
    u->url = estrdup(url);
    if(spliturl(u->url, &su) < 0){
    Fail:
        freeurl(u);
        return nil;
    }

    /* RFC2396 sec 3.1 says relative URIs are distinguished by absent scheme */
    if(su.scheme.s==nil){
```

```
        if(urldebug)
            fprint(2, "parseurl has nil scheme\n");
        if(resolve_relative(&su, base, u) < 0 || spliturl(u->url, &su) < 0)
            goto Fail;
        if(u->ischeme == UScurrent){
            /* 'u.url' refers to current document; set fragment and return */
            if(parse_fragment(&su, u) < 0)
                goto Fail;
            return u;
        }
    }

    if(parse_scheme(&su, u) < 0
    || parse_fragment(&su, u) < 0)
        goto Fail;

    if(u->ischeme == USunknown){
        if(parse_unknown_part(&su, u) < 0)
            goto Fail;
        return u;
    }

    if(parse_query(&su, u) < 0
    || parse_authority(&su, u) < 0
    || parse_abspath(&su, u) < 0)
        goto Fail;

    if(u->ischeme < nelem(postparse) && postparse[u->ischeme])
        if((*postparse[u->ischeme])(u) < 0)
            goto Fail;

    setmalloctag(u, getcallerpc(&url));
    return u;
}
```

⟨*function* `freeurl` *(webfs)* 124⟩≡ (127b)

```
void
freeurl(Url *u)
{
    if(u == nil)
        return;
    free(u->url);
    free(u->scheme);
    free(u->schemedata);
    free(u->authority);
    free(u->user);
    free(u->passwd);
    free(u->host);
    free(u->port);
    free(u->path);
    free(u->query);
    free(u->fragment);
    switch(u->ischeme){
    case UShttp:
        free(u->http.page_spec);
        break;
    case USftp:
        free(u->ftp.path_spec);
        free(u->ftp.type);
        break;
```

```
    }
    free(u);
}
```

⟨*function* `rewriteurl` *(webfs)* 125a⟩≡ (127b)

```
void
rewriteurl(Url *u)
{
    char *s;

    if(u->schemedata)
        s = estrmanydup(u->scheme, ":", u->schemedata, nil);
    else
        s = estrmanydup(u->scheme, "://",
            u->user ? u->user : "",
            u->passwd ? ":" : "", u->passwd ? u->passwd : "",
            u->user ? "@" : "", u->host ? u->host : "",
            u->port ? ":" : "", u->port ? u->port : "",
            u->path,
            u->query ? "?" : "", u->query ? u->query : "",
            u->fragment ? "#" : "", u->fragment ? u->fragment : "",
            nil);
    free(u->url);
    u->url = s;
}
```

⟨*function* `seturlquery` *(webfs)* 125b⟩≡ (127b)

```
int
seturlquery(Url *u, char *query)
{
    if(query == nil){
        free(u->query);
        u->query = nil;
        return 0;
    }

    if(!ismatch(REquery, query, "query"))
        return -1;

    free(u->query);
    u->query = estrdup(query);
    return 0;
}
```

⟨*function* `dupp` *(webfs)* 125c⟩≡ (127b)

```
static void
dupp(char **p)
{
    if(*p)
        *p = estrdup(*p);
}
```

⟨*function* `copyurl` *(webfs)* 125d⟩≡ (127b)

```
Url*
copyurl(Url *u)
{
    Url *v;

    v = emalloc(sizeof(Url));
    *v = *u;
```

```
    dupp(&v->url);
    dupp(&v->scheme);
    dupp(&v->schemedata);
    dupp(&v->authority);
    dupp(&v->user);
    dupp(&v->passwd);
    dupp(&v->host);
    dupp(&v->port);
    dupp(&v->path);
    dupp(&v->query);
    dupp(&v->fragment);

    switch(v->ischeme){
    case UShttp:
        dupp(&v->http.page_spec);
        break;
    case USftp:
        dupp(&v->ftp.path_spec);
        dupp(&v->ftp.type);
        break;
    }
    return v;
}
```

⟨*function* dhex*(webfs)* 126a⟩≡ (127b)

```
static int
dhex(char c)
{
    if('0' <= c && c <= '9')
        return c-'0';
    if('a' <= c && c <= 'f')
        return c-'a'+10;
    if('A' <= c && c <= 'F')
        return c-'A'+10;
    return 0;
}
```

⟨*function* escapeurl*(webfs)* 126b⟩≡ (127b)

```
char*
escapeurl(char *s, int (*needesc)(int))
{
    int n;
    char *t, *u;
    Rune r;
    static char *hex = "0123456789abcdef";

    n = 0;
    for(t=s; *t; t++)
        if((*needesc)(*t))
            n++;

    u = emalloc(strlen(s)+2*n+1);
    t = u;
    for(; *s; s++){
        s += chartorune(&r, s);
        if(r >= 0xFF){
            werrstr("URLs cannot contain Runes > 0xFF");
            free(t);
            return nil;
        }
```

```
        if((*needesc)(r)){
            *u++ = '%';
            *u++ = hex[(r>>4)&0xF];
            *u++ = hex[r&0xF];
        }else
            *u++ = r;
    }
    *u = '\0';
    return t;
}
```

⟨*function* unescapeurl*(webfs)* 127a⟩≡ (127b)

```
char*
unescapeurl(char *s)
{
    char *r, *w;
    Rune rune;

    s = estrdup(s);
    for(r=w=s; *r; r++){
        if(*r=='%'){
            r++;
            if(!isxdigit(r[0]) || !isxdigit(r[1])){
                werrstr("bad escape sequence '%.3s' in URL", r);
                return nil;
            }
            if(r[0]=='0' && r[2]=='0'){
                werrstr("escaped NUL in URL");
                return nil;
            }
            rune = (dhex(r[0])<<4)|dhex(r[1]);  /* latin1 */
            w += runetochar(w, &rune);
            r += 2;
        }else
            *w++ = *r;
    }
    *w = '\0';
    return s;
}
```

⟨webfs/url.c 127b⟩≡

```
/*
 * This is a URL parser, written to parse "Common Internet Scheme" URL
 * syntax as described in RFC1738 and updated by RFC2396.  Only absolute URLs
 * are supported, using "server-based" naming authorities in the schemes.
 * Support for literal IPv6 addresses is included, per RFC2732.
 *
 * Current "known" schemes: http, ftp, file.
 *
 * We can do all the parsing operations without Runes since URLs are
 * defined to be composed of US-ASCII printable characters.
 * See RFC1738, RFC2396.
 */

#include <u.h>
#include <libc.h>
#include <ctype.h>
#include <regexp.h>
#include <plumb.h>
#include <thread.h>
```

```
#include <fcall.h>
#include <9p.h>
#include "dat.h"
#include "fns.h"

int urldebug;

/* If set, relative paths with leading ".." segments will have them trimmed */
⟨constant RemoveExtraRelDotDots(webfs) 111b⟩
⟨constant ExpandCurrentDocUrls(webfs) 111c⟩

⟨global schemestrtab(webfs) 110a⟩

⟨function ischeme(webfs) 111d⟩

/*
 * URI splitting regexp is from RFC2396, Appendix B:
 *              ^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
 *               12            3  4          5       6  7        8 9
 *
 * Example: "http://www.ics.uci.edu/pub/ietf/uri/#Related"
 * $2 = scheme                  "http"
 * $4 = authority               "www.ics.uci.edu"
 * $5 = path                    "/pub/ietf/uri/"
 * $7 = query                   <undefined>
 * $9 = fragment                "Related"
 */

/*
 * RFC2396, Sec 3.1, contains:
 *
 * Scheme names consist of a sequence of characters beginning with a
 * lower case letter and followed by any combination of lower case
 * letters, digits, plus ("+"), period ("."), or hyphen ("-").  For
 * resiliency, programs interpreting URI should treat upper case letters
 * as equivalent to lower case in scheme names (e.g., allow "HTTP" as
 * well as "http").
 */

/*
 * For server-based naming authorities (RFC2396 Sec 3.2.2):
 *    server        = [ [ userinfo "@" ] hostport ]
 *    userinfo      = *( unreserved | escaped |
 *                       ";" | ":" | "&" | "=" | "+" | "$" | "," )
 *    hostport      = host [ ":" port ]
 *    host          = hostname | IPv4address
 *    hostname      = *( domainlabel "." ) toplabel [ "." ]
 *    domainlabel   = alphanum | alphanum *( alphanum | "-" ) alphanum
 *    toplabel      = alpha | alpha *( alphanum | "-" ) alphanum
 *    IPv4address   = 1*digit "." 1*digit "." 1*digit "." 1*digit
 *    port          = *digit
 *
 *  The host is a domain name of a network host, or its IPv4 address as a
 *  set of four decimal digit groups separated by ".".  Literal IPv6
 *  addresses are not supported.
 *
 * Note that literal IPv6 address support is outlined in RFC2732:
 *    host          = hostname | IPv4address | IPv6reference
 *    ipv6reference = "[" IPv6address "]"               (RFC2373)
 *
```

```
 * Since hostnames and numbers will have to be resolved by the OS anyway,
 * we don't have to parse them too pedantically (counting '.'s, checking
 * for well-formed literal IP addresses, etc.).
 *
 * In FTP/file paths, we reject most ";param"s and querys.  In HTTP paths,
 * we just pass them through.
 *
 * Instead of letting a "path" be 0-or-more characters as RFC2396 suggests,
 * we'll say it's 1-or-more characters, 0-or-1 times.  This way, an absent
 * path yields a nil substring match, instead of an empty one.
 *
 * We're more restrictive than RFC2396 indicates with "userinfo" strings,
 * insisting they have the form "[user[:password]]".  This may need to
 * change at some point, however.
 */

/* RE character-class components -- these go in brackets */
⟨constant PUNCT (webfs) 112a⟩
⟨constant RES (webfs) 112b⟩
⟨constant ALNUM (webfs) 112c⟩
⟨constant HEX (webfs) 112d⟩
⟨constant UNRES (webfs) 112e⟩

/* RE components; _N => has N parenthesized subexpressions when expanded */
⟨constant ESCAPED_1 (webfs) 112f⟩
⟨constant URIC_2 (webfs) 112g⟩
⟨constant URICNOSLASH_2 (webfs) 112h⟩
⟨constant USERINFO_2 (webfs) 112i⟩
⟨constant PCHAR_2 (webfs) 112j⟩
⟨constant PSEGCHAR_3 (webfs) 112k⟩

typedef struct Retab Retab;
⟨struct Retab (webfs) 110b⟩

⟨enum RExxx (webfs) 110c⟩

⟨global retab (webfs) 110d⟩


⟨function countleftparen (webfs) 112l⟩

⟨function initurl (webfs) 113a⟩

typedef struct SplitUrl SplitUrl;
⟨struct SplitUrl (webfs) 111a⟩

⟨function merge_relative_path (webfs) 113b⟩

⟨function resolve_relative (webfs) 114⟩

⟨function regx (webfs) 117a⟩

⟨function ismatch (webfs) 117b⟩

⟨function spliturl (webfs) 117c⟩

⟨function parse_scheme (webfs) 118⟩

⟨function parse_unknown_part (webfs) 119a⟩
```

⟨*function* `parse_userinfo` *(webfs)* 119b⟩

⟨*function* `parse_host` *(webfs)* 120a⟩

⟨*function* `parse_authority` *(webfs)* 120b⟩

⟨*function* `parse_abspath` *(webfs)* 121a⟩

⟨*function* `parse_query` *(webfs)* 121b⟩

⟨*function* `parse_fragment` *(webfs)* 121c⟩

⟨*function* `postparse_http` *(webfs)* 121d⟩

⟨*function* `postparse_ftp` *(webfs)* 122⟩

⟨*function* `postparse_file` *(webfs)* 123a⟩

```
static int (*postparse[])(Url*) = {
    nil,
    postparse_http,
    postparse_http,
    postparse_ftp,
    postparse_file,
};
```

⟨*function* `parseurl` *(webfs)* 123b⟩

⟨*function* `freeurl` *(webfs)* 124⟩

⟨*function* `rewriteurl` *(webfs)* 125a⟩

⟨*function* `seturlquery` *(webfs)* 125b⟩

⟨*function* `dupp` *(webfs)* 125c⟩

⟨*function* `copyurl` *(webfs)* 125d⟩

⟨*function* `dhex` *(webfs)* 126a⟩

⟨*function* `escapeurl` *(webfs)* 126b⟩

⟨*function* `unescapeurl` *(webfs)* 127a⟩

webfs/util.c

⟨*function* `erealloc` *(webfs)* 130a⟩≡ (132b)

```
void*
erealloc(void *a, uint n)
{
    a = realloc(a, n);
    if(a == nil)
        sysfatal("realloc %d: out of memory", n);
    setrealloctag(a, getcallerpc(&a));
    return a;
}
```

⟨*function* `emalloc` *(webfs)* 130b⟩≡ (132b)

```
void*
emalloc(uint n)
{
    void *a;

    a = mallocz(n, 1);
    if(a == nil)
        sysfatal("malloc %d: out of memory", n);
    setmalloctag(a, getcallerpc(&n));
    return a;
}
```

⟨*function* `estrdup` *(webfs)* 131a⟩≡ (132b)

```
char*
estrdup(char *s)
{
    s = strdup(s);
    if(s == nil)
        sysfatal("strdup: out of memory");
    setmalloctag(s, getcallerpc(&s));
    return s;
}
```

⟨*function* `estredup` *(webfs)* 131b⟩≡ (132b)

```
char*
estredup(char *s, char *e)
{
    char *t;

    t = emalloc(e-s+1);
    memmove(t, s, e-s);
    t[e-s] = '\0';
    setmalloctag(t, getcallerpc(&s));
    return t;
}
```

⟨*function* `estrmanydup` *(webfs)* 131c⟩≡ (132b)

```
char*
estrmanydup(char *s, ...)
{
    char *p, *t;
    int len;
    va_list arg;

    len = strlen(s);
    va_start(arg, s);
    while((p = va_arg(arg, char*)) != nil)
        len += strlen(p);
    len++;

    t = emalloc(len);
    strcpy(t, s);
    va_start(arg, s);
    while((p = va_arg(arg, char*)) != nil)
        strcat(t, p);
    return t;
}
```

⟨*function* `strlower` *(webfs)* 132a⟩≡ (132b)

```
char*
strlower(char *s)
{
    char *t;

    for(t=s; *t; t++)
        if('A' <= *t && *t <= 'Z')
            *t += 'a'-'A';
    return s;
}
```

⟨`webfs/util.c` 132b⟩≡

```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ndb.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>
#include <ctype.h>
#include "dat.h"
#include "fns.h"
```

⟨*function* `erealloc` *(webfs)* 130a⟩

⟨*function* `emalloc` *(webfs)* 130b⟩

⟨*function* `estrdup` *(webfs)* 131a⟩

⟨*function* `estredup` *(webfs)* 131b⟩

⟨*function* `estrmanydup` *(webfs)* 131c⟩

⟨*function* `strlower` *(webfs)* 132a⟩

E.2 mothra/

mothra/html.h

⟨*struct* `Pair` *(mothra)* 132c⟩≡ (136i)

```
struct Pair{
    char *name;
    char *value;
};
```

⟨*struct* `Entity` *(mothra)* 132d⟩≡ (136i)

```
struct Entity{
    char *name;
    Rune value;
};
```

⟨*struct* `Tag` *(mothra)* 132e⟩≡ (136i)

```
struct Tag{
    char *name;
    int action;
};
```

⟨*struct* Stack*(mothra)* 133a⟩≡ (136i)

```
struct Stack{
    int tag;            /* html tag being processed */
    int pre;            /* in preformatted text? */
    int font;           /* typeface */
    int size;           /* point size of text */
    int sub;            /* < 0 superscript, > 0 subscript */
    int margin;         /* left margin position */
    int indent;         /* extra indent at paragraph start */
    int number;         /* paragraph number */
    int ismap;          /* flag of <img> */
    int isscript;               /* inside <script> */
    int strike;         /* flag of <strike> */
    int width;          /* size of image */
    int height;

    char *image;                /* arg of <img> */
    char *link;         /* arg of <a href=...> */
    char *name;         /* arg of <a name=...> */
};
```

⟨*struct* Hglob*(mothra)* 133b⟩≡ (136i)

```
/*
 * Globals -- these are packed up into a struct that gets passed around
 * so that multiple parsers can run concurrently
 */
struct Hglob{
    char *tp;           /* pointer in text buffer */
    char *name;         /* input file name */
    int hfd;            /* input file descriptor */
    char hbuf[NHBUF];   /* input buffer */
    char *hbufp;                /* next character in buffer */
    char *ehbuf;                /* end of good characters in buffer */
    int heof;           /* end of file flag */
    int peekc[NPEEKC];  /* characters to re-read */
    int npeekc;         /* # of characters to re-read */
    char token[NTOKEN]; /* if token type is TEXT */
    Pair attr[NATTR];   /* tag attribute/value pairs */
    int nsp;            /* # of white-space characters before TEXT token */
    int spacc;          /* place to accumulate more spaces */
                /* if negative, won't accumulate! */
    int tag;            /* if token type is TAG or END */

    Stack stack[NSTACK];        /* parse stack */
    Stack *state;               /* parse stack pointer */

    int lineno;         /* input line number */
    int linebrk;                /* flag set if we require a line-break in output */
    int para;           /* flag set if we need an indent at the break */
    char *text;         /* text buffer */
    char *etext;                /* end of text buffer */
    Form *form;         /* data for form under construction */

    Www *dst;           /* where the text goes */
};
```

⟨*enum* Txx*(mothra)* 133c⟩≡ (136i)

```
/*
 * Token types
 */
```

```
enum{
    TAG=1,
    ENDTAG,
    TEXT,
};
```

⟨*enum* XFonts*(mothra)* 134a⟩≡ (136i)

```
/*
 * fonts
 */
enum{
    ROMAN,
    ITALIC,
    BOLD,
    CWIDTH,
};
```

⟨*enum* XFontSize*(mothra)* 134b⟩≡ (136i)

```
/*
 * font sizes
 */
enum{
    SMALL,
    NORMAL,
    LARGE,
    ENORMOUS,
};
```

⟨*enum* XLenDir*(mothra)* 134c⟩≡ (136i)

```
/*
 * length direction
 */
enum{
    HORIZ,
    VERT,
};
```

⟨*enum* Tagxxx*(mothra)* 134d⟩≡ (136i)

```
/*
 * Token names for the html parser.
 * Tag_end corresponds to </end> tags.
 * Tag_text tags text not in a tag.
 * Those two must follow the others.
 */
enum{
    Tag_comment,

    Tag_a,
    Tag_abbr,
    Tag_acronym,
    Tag_address,
    Tag_applet,
    Tag_article,
    Tag_audio,
    Tag_b,
    Tag_base,
    Tag_blockquot,
    Tag_body,
    Tag_br,
    Tag_button,
```

```
Tag_center,
Tag_cite,
Tag_code,
Tag_dd,
Tag_del,
Tag_div,
Tag_dfn,
Tag_dir,
Tag_dl,
Tag_dt,
Tag_em,
Tag_embed,
Tag_figure,
Tag_figcaption,
Tag_font,
Tag_form,
Tag_frame,  /* rm 5.8.97 */
Tag_h1,
Tag_h2,
Tag_h3,
Tag_h4,
Tag_h5,
Tag_h6,
Tag_head,
Tag_hr,
Tag_html,
Tag_i,
Tag_iframe,
Tag_img,
Tag_image,
Tag_input,
Tag_ins,
Tag_isindex,
Tag_kbd,
Tag_key,
Tag_li,
Tag_link,
Tag_listing,
Tag_menu,
Tag_meta,
Tag_nextid,
Tag_object,
Tag_ol,
Tag_option,
Tag_p,
Tag_plaintext,
Tag_pre,
Tag_s,
Tag_samp,
Tag_script,
Tag_select,
Tag_span,
Tag_strike,
Tag_strong,
Tag_style,
Tag_sub,
Tag_sup,
Tag_source,
Tag_table,  /* rm 3.8.00 */
Tag_td,
```

```
    Tag_th,
    Tag_textarea,
    Tag_title,
    Tag_tr,
    Tag_tt,
    Tag_u,
    Tag_ul,
    Tag_var,
    Tag_video,
    Tag_wbr,
    Tag_xmp,

    Tag_end,    /* also used to indicate unrecognized start tag */
    Tag_text,
};
```

⟨*constant* NSTACK*(mothra)* 136a⟩≡ (136i)
```
#define NSTACK  100     /* html grammar is not recursive, so 30 or so should do */
```

⟨*constant* NHBUF*(mothra)* 136b⟩≡ (136i)
```
#define NHBUF   8192 //IOUNIT   /* Input buffer size */
```

⟨*constant* NPEEKC*(mothra)* 136c⟩≡ (136i)
```
#define NPEEKC  3       /* Maximum lookahead */
```

⟨*constant* NTOKEN*(mothra)* 136d⟩≡ (136i)
```
#define NTOKEN  65536   /* Maximum token length */
```

⟨*constant* NATTR*(mothra)* 136e⟩≡ (136i)
```
#define NATTR   512     /* Maximum number of attributes of a tag */
```

⟨*constant* STAG*(mothra)* 136f⟩≡ (136i)
```
#define STAG    65536
```

⟨*constant* ETAG*(mothra)* 136g⟩≡ (136i)
```
#define ETAG    65537
```

⟨*constant* EOF*(mothra)* 136h⟩≡ (136i)
```
#define EOF     -1
```

⟨mothra/html.h 136i⟩≡
```
/*
 * Parameters
 */
```
⟨*constant* NSTACK*(mothra)* 136a⟩
⟨*constant* NHBUF*(mothra)* 136b⟩
⟨*constant* NPEEKC*(mothra)* 136c⟩
⟨*constant* NTOKEN*(mothra)* 136d⟩
⟨*constant* NATTR*(mothra)* 136e⟩
```
typedef struct Pair Pair;
typedef struct Tag Tag;
typedef struct Stack Stack;
typedef struct Hglob Hglob;
typedef struct Form Form;
typedef struct Entity Entity;
```

⟨*struct* Pair*(mothra)* 132c⟩

⟨*struct* Entity*(mothra)* 132d⟩

⟨*struct* `Tag`*(mothra)* 132e⟩

⟨*struct* `Stack`*(mothra)* 133a⟩

⟨*struct* `Hglob`*(mothra)* 133b⟩

⟨*enum* `Txx`*(mothra)* 133c⟩

```
/*
 * Magic characters corresponding to
 *      literal < followed by / ! or alpha,
 *      literal > and
 *      end of file
 */
```
⟨*constant* `STAG`*(mothra)* 136f⟩
⟨*constant* `ETAG`*(mothra)* 136g⟩
⟨*constant* `EOF`*(mothra)* 136h⟩

⟨*enum* `XFonts`*(mothra)* 134a⟩

⟨*enum* `XFontSize`*(mothra)* 134b⟩

⟨*enum* `XLenDir`*(mothra)* 134c⟩

```
int strtolength(Hglob *g, int dir, char *str);
```

⟨*enum* `Tagxxx`*(mothra)* 134d⟩

```
enum{
    NTAG=Tag_end,
    END=1,      /* tag must have a matching end tag */
    NOEND,      /* tag must not have a matching end tag */
    OPTEND,     /* tag may have a matching end tag */
    ERR,                /* tag must not occur */
};

Tag tag[];
void rdform(Hglob *);
void endform(Hglob *);
char *pl_getattr(Pair *, char *);
int pl_hasattr(Pair *, char *);
void pl_htmloutput(Hglob *, int, char *, Field *);

#pragma incomplete Form
#pragma incomplete Field
```

mothra/mothra.h

⟨*struct* `Action`*(mothra)* 137⟩≡ (139)

```
struct Action{
    char *image;
    Field *field;
    char *link;
    char *name;
    int ismap;
    int width;
```

```
    int height;
};
```

⟨*struct* Url*(mothra)* 138a⟩≡ (139)

```
struct Url{
    char *basename;
    char *reltext;
    char fullname[NNAME];
    char tag[NNAME];
    char contenttype[NNAME];
    int map;            /* is this an image map? */
};
```

⟨*struct* Www*(mothra)* 138b⟩≡ (139)

```
struct Www{
    Url *url;
    void *pix;
    void *form;
    char title[NTITLE];
    Rtext *text;
    int yoffs;
    int gottitle;               /* title got drawn */
    bool changed;               /* reader sets this every time it updates page */
    int finished;               /* reader sets this when done */
    int alldone;                /* page will not change further -- used to adjust cursor */
};
```

⟨*enum* XKind*(mothra)* 138c⟩≡ (139)

```
enum{
    PLAIN,
    HTML,

    GIF,
    JPEG,
    PNG,
    BMP,
    ICO,

    PAGE,
};
```

⟨*enum* Axxx*(mothra)* 138d⟩≡ (139)

```
/*
 *  authentication types
 */
enum{
    ANONE,
    ABASIC,
};
```

⟨*enum* HttpMethod*(mothra)* 138e⟩≡ (139)

```
/*
 * HTTP methods
 */
enum{
    GET=1,
    POST,
};
```

⟨mothra/mothra.h 139⟩≡

```
enum{
    NWWW=64,    /* # of pages we hold in the log */
    NXPROC=5,   /* # of parallel procs loading the pix */
    NPIXMB=8,   /* megabytes of image data to keep arround */
    NNAME=512,
    NLINE=256,
    NAUTH=128,
    NTITLE=81,  /* length of title (including nul at end) */
    NLABEL=50,  /* length of option name in forms */
    NREDIR=10,  /* # of redirections we'll tolerate before declaring a loop */
};

typedef struct Action Action;
typedef struct Url Url;
typedef struct Www Www;
typedef struct Field Field;
```

⟨*struct* Action*(mothra)* 137⟩

⟨*struct* Url*(mothra)* 138a⟩

⟨*struct* Www*(mothra)* 138b⟩

⟨*enum* XKind*(mothra)* 138c⟩

⟨*enum* Axxx*(mothra)* 138d⟩

```
Image *hrule, *bullet, *linespace;
int chrwidth;           /* nominal width of characters in font */
Panel *text;            /* Panel displaying the current www page */

bool debug;             /* command line flag */
```

⟨*enum* HttpMethod*(mothra)* 138e⟩

```
void finish(Www *w);
void plrdhtml(char *, int, Www *, int);
void plrdplain(char *, int, Www *);
void htmlerror(char *, int, char *, ...);       /* user-supplied routine */
void seturl(Url *, char *, char *);
void freeurl(Url *);
Url *selurl(char *);
void getpix(Rtext *, Www *);
ulong countpix(void *p);
void freepix(void *p);
void dupfds(int fd, ...);
int pipeline(int fd, char *fmt, ...);
void getfonts(void);
void *emalloc(int);
void nstrcpy(char *to, char *from, int len);
void freeform(void *p);
int Ufmt(Fmt *f);
#pragma varargck type "U" char*
void message(char *, ...);
int filetype(int, char *, int);
int mimetotype(char *);
int snooptype(int);
void mkfieldpanel(Rtext *);
```

```
void geturl(char *, int, int, int);
char *urlstr(Url *);
int urlpost(Url*, char*);
int urlget(Url*, int);
int urlresolve(Url *);

Mouse mouse;
```

mothra/forms.c

⟨*struct* Form*(mothra)* 140a⟩≡ (153b)

```
struct Form{
    int method;
    char *ctype;
    char *action;
    Field *fields, *efields;
    Form *next;
};
```

⟨*struct* Field*(mothra)* 140b⟩≡ (153b)

```
struct Field{
    Field *next;
    Form *form;
    char *name;
    char *value;
    int checked;
    int size;           /* should be a point, but that feature is deprecated */
    int maxlength;
    int type;
    int rows, cols;
    Option *options;
    int multiple;
    int state;          /* is the button marked? */
    Panel *p;
    Panel *pulldown;
    Panel *textwin;
};
```

⟨*enum* FieldType*(mothra)* 140c⟩≡ (153b)

```
/*
 * Field types
 */
enum{
    TYPEIN=1,
    CHECK,
    PASSWD,
    RADIO,
    SUBMIT,
    RESET,
    BUTTON,
    SELECT,
    TEXTWIN,
    HIDDEN,
    INDEX,
    FILE,
};
```

⟨*struct* `Option`*(mothra)* 141a⟩≡ (153b)
```
struct Option{
    int selected;
    int def;
    char label[NLABEL+1];
    char *value;
    Option *next;
};
```

⟨*constant* `BOUNDARY`*(mothra)* 141b⟩≡ (153b)
```
#define BOUNDARY "nAboJ9uN6ZXsqoVGzLAdjKq97TWDTGjo"
```

⟨*function* `newfield`*(mothra)* 141c⟩≡ (153b)
```
Field *newfield(Form *form){
    Field *f;
    f=emalloc(sizeof(Field));
    if(form->efields==0)
        form->fields=f;
    else
        form->efields->next=f;
    form->efields=f;
    f->next=0;
    f->form=form;
    return f;
}
```

⟨*function* `rdform`*(mothra)* 141d⟩≡ (153b)
```
/*
 * Called by rdhtml on seeing a forms-related tag
 */
void rdform(Hglob *g){
    char *s;
    Field *f;
    Option *o, **op;
    Form *form;
    switch(g->tag){
    default:
        fprint(2, "Bad tag <%s> in rdform (Can't happen!)\n", g->token);
        return;
    case Tag_form:
        if(g->form){
            htmlerror(g->name, g->lineno, "nested forms illegal\n");
            break;
        }
        g->form=emalloc(sizeof(Form));
        s=pl_getattr(g->attr, "action");
        g->form->action=strdup((s && *s) ? s : g->dst->url->fullname);
        s=pl_getattr(g->attr, "method");
        if(s==0 || *s==0)
            g->form->method=GET;
        else if(cistrcmp(s, "post")==0)
            g->form->method=POST;
        else{
            if(cistrcmp(s, "get")!=0)
                htmlerror(g->name, g->lineno,
                    "unknown form method %s\n", s);
            g->form->method=GET;
        }
        s=pl_getattr(g->attr, "enctype");
        if(s && cistrcmp(s, "multipart/form-data")==0)
```

```
        g->form->ctype = "multipart/form-data; boundary=" BOUNDARY;
    g->form->fields=0;

    g->form->next = g->dst->form;
    g->dst->form = g->form;
    break;
case Tag_input:
case Tag_button:
    if(g->form==0){
        /* no form, assume link button */
        form = emalloc(sizeof(Form));
        form->method = 0;
        form->fields = 0;
        form->efields = 0;
        if(g->state->link)
            form->action = strdup(g->state->link);
        form->next = g->dst->form;
        g->dst->form = form;
        f=newfield(form);
    } else
        f=newfield(g->form);
    s=pl_getattr(g->attr, "name");
    if(s==0 || *s == 0)
        f->name=0;
    else
        f->name=strdup(s);
    s=pl_getattr(g->attr, "value");
    if(s==0)
        f->value=strdup("");
    else
        f->value=strdup(s);
    f->checked=pl_hasattr(g->attr, "checked");
    s=pl_getattr(g->attr, "size");
    if(s==0 || *s==0)
        f->size=20;
    else
        f->size=atoi(s);
    s=pl_getattr(g->attr, "maxlength");
    if(s==0 || *s==0)
        f->maxlength=0x3fffffff;
    else
        f->maxlength=atoi(s);
    s=pl_getattr(g->attr, "type");
    if((g->tag == Tag_button) &&
       (s==0 || cistrcmp(s, "reset") || cistrcmp(s, "button")))
        s="submit";
    else if(s==0)
        s="text";
    if(cistrcmp(s, "checkbox")==0)
        f->type=CHECK;
    else if(cistrcmp(s, "radio")==0)
        f->type=RADIO;
    else if(cistrcmp(s, "submit")==0)
        f->type=SUBMIT;
    else if(cistrcmp(s, "image")==0){
        f->type=SUBMIT;
        s=pl_getattr(g->attr, "src");
        if(s && *s){
            free(g->state->image);
            g->state->image = strdup(s);
```

```
        }
        s=pl_getattr(g->attr, "width");
        if(s && *s)
            g->state->width=strtolength(g, HORIZ, s);
        s=pl_getattr(g->attr, "height");
        if(s && *s)
            g->state->height=strtolength(g, VERT, s);
        s=pl_getattr(g->attr, "alt");
        if(s==0 || *s == 0) s = f->value;
        pl_htmloutput(g, g->nsp, s, f);
        free(g->state->image);
        g->state->image = 0;
        g->state->width=0;
        g->state->height=0;
        break;
    }
    else if(cistrcmp(s, "button")==0)
        f->type=BUTTON;
    else if(cistrcmp(s, "file")==0)
        f->type=FILE;
    else if(cistrcmp(s, "reset")==0)
        f->type=RESET;
    else if(cistrcmp(s, "hidden")==0)
        f->type=HIDDEN;
    else{
        f->type=TYPEIN;
        if(cistrcmp(s, "password")==0)
            f->type=PASSWD;
        s=f->name;
        if(s && cistrcmp(s, "isindex")==0)
            f->type=INDEX;

        /*
         * If there's exactly one attribute, use its value as the name,
         * regardless of the attribute name.  This makes
         * http://linus.att.com/ias/puborder.html work.
         */
        if(s==0){
            if(g->attr[0].name && g->attr[1].name==0)
                f->name=strdup(g->attr[0].value);
            else
                f->name=strdup("no-name");
        }
    }
    if((f->type==CHECK || f->type==RADIO) && !pl_hasattr(g->attr, "value")){
        free(f->value);
        f->value=strdup("on");
    }
    if(f->type!=HIDDEN)
        pl_htmloutput(g, g->nsp, f->value[0]?f->value:"blank field", f);
    break;
case Tag_select:
    if(g->form==0){
    BadTag:
        htmlerror(g->name, g->lineno, "<%s> not in form, ignored\n",
            tag[g->tag].name);
        break;
    }
    f=newfield(g->form);
    s=pl_getattr(g->attr, "name");
```

```
    if(s==0 || *s==0){
        f->name=strdup("select");
        htmlerror(g->name, g->lineno, "select has no name=\n");
    }
    else
        f->name=strdup(s);
    f->multiple=pl_hasattr(g->attr, "multiple");
    f->type=SELECT;
    f->size=0;
    f->options=0;
    g->text=g->token;
    g->tp=g->text;
    g->etext=g->text;
    break;
case Tag_option:
    if(g->form==0) goto BadTag;
    if((f=g->form->efields)==0) goto BadTag;
    if(f->size<8)
        f->size++;
    o=emalloc(sizeof(Option));
    for(op=&f->options;*op;op=&(*op)->next);
    *op=o;
    o->next=0;
    g->text=o->label;
    g->tp=o->label;
    g->etext=o->label+NLABEL;
    memset(o->label, 0, NLABEL+1);
    *g->tp++=' ';
    o->def=pl_hasattr(g->attr, "selected");
    o->selected=o->def;
    if(pl_hasattr(g->attr, "disabled"))
        o->selected=0;
    s=pl_getattr(g->attr, "value");
    if(s==0)
        o->value=o->label+1;
    else
        o->value=strdup(s);
    break;
case Tag_textarea:
    if(g->form==0) goto BadTag;
    f=newfield(g->form);
    s=pl_getattr(g->attr, "name");
    if(s==0 || *s==0){
        f->name=strdup("enter text");
        htmlerror(g->name, g->lineno, "select has no name=\n");
    }
    else
        f->name=strdup(s);
    s=pl_getattr(g->attr, "rows");
    f->rows=(s && *s)?atoi(s):8;
    s=pl_getattr(g->attr, "cols");
    f->cols=(s && *s)?atoi(s):30;
    f->type=TEXTWIN;
    /* suck up initial text */
    pl_htmloutput(g, g->nsp, f->name, f);
    break;
case Tag_isindex:
    /*
     * Make up a form with one tag, of type INDEX
     * I have seen a page with <ISINDEX PROMPT="Enter a title here ">,
```

```
         * which is nonstandard and not handled here.
         */
        form=emalloc(sizeof(Form));
        form->fields=0;
        form->efields=0;
        s=pl_getattr(g->attr, "action");
        form->action=strdup((s && *s) ? s : g->dst->url->fullname);
        form->method=GET;
        form->fields=0;
        f=newfield(form);
        f->name=0;
        f->value=strdup("");
        f->size=20;
        f->maxlength=0x3fffffff;
        f->type=INDEX;
        pl_htmloutput(g, g->nsp, f->value[0]?f->value:"blank field", f);
        break;
    }
}
```

⟨*function* endform*(mothra)* 145a⟩≡ (153b)

```
/*
 * Called by rdhtml on seeing a forms-related end tag
 */
void endform(Hglob *g){
    Field *f;

    switch(g->tag){
    case Tag_form:
        g->form=0;
        break;
    case Tag_select:
        if(g->form==0)
            htmlerror(g->name, g->lineno, "</select> not in form, ignored\n");
        else if((f=g->form->efields)==0)
            htmlerror(g->name, g->lineno, "spurious </select>\n");
        else
            pl_htmloutput(g, g->nsp, f->name, f);
        break;
    case Tag_textarea:
        break;
    }
}
```

⟨*function* nullgen*(mothra)* 145b⟩≡ (153b)

```
char *nullgen(Panel *, int ){
    return 0;
}
```

⟨*function* selgen*(mothra)* 145c⟩≡ (153b)

```
char *selgen(Panel *p, int index){
    Option *a;
    Field *f;
    f=p->userp;
    if(f==0) return 0;
    for(a=f->options;index!=0 && a!=0;--index,a=a->next);
    if(a==0) return 0;
    a->label[0]=a->selected?'*':' ';
    return a->label;
}
```

⟨*function* seloption*(mothra)* 146a⟩≡ (153b)

```
char *seloption(Field *f){
    Option *a;
    for(a=f->options;a!=0;a=a->next)
        if(a->selected)
            return a->label+1;
    return f->name;
}
```

⟨*function* mkfieldpanel*(mothra)* 146b⟩≡ (153b)

```
void mkfieldpanel(Rtext *t){
    Action *a;
    Panel *win, *scrl;
    Field *f;

    if((a = t->user) == nil)
        return;
    if((f = a->field) == nil)
        return;

    f->p=0;
    switch(f->type){
    case TYPEIN:
        f->p=plentry(0, 0, f->size*chrwidth, f->value, h_submittype);
        break;
    case PASSWD:
        f->p=plentry(0, USERFL, f->size*chrwidth, f->value, h_submittype);
        break;
    case CHECK:
        f->p=plcheckbutton(0, 0, "", h_checkinput);
        f->state=f->checked;
        plsetbutton(f->p, f->checked);
        break;
    case RADIO:
        f->p=plradiobutton(0, 0, "", h_radioinput);
        f->state=f->checked;
        plsetbutton(f->p, f->checked);
        break;
    case SUBMIT:
        f->p=plbutton(0, 0, f->value[0]?f->value:"submit", h_submitinput);
        break;
    case RESET:
        f->p=plbutton(0, 0, f->value[0]?f->value:"reset", h_resetinput);
        break;
    case BUTTON:
        f->p=plbutton(0, 0, f->value[0]?f->value:"button", h_buttoninput);
        break;
    case FILE:
        f->p=plbutton(0, 0, f->value[0]?f->value:"file", h_fileinput);
        break;
    case SELECT:
        if(f->size <= 0)
            f->size=1;
        f->pulldown=plgroup(0,0);
        scrl=plscrollbar(f->pulldown, PACKW|FILLY);
        win=pllist(f->pulldown, PACKN, nullgen, f->size, h_select);
        win->userp=f;
        plinitlist(win, PACKN, selgen, f->size, h_select);
        plscroll(win, 0, scrl);
        plpack(f->pulldown, Rect(0,0,1024,1024));
```

```
        f->p=plpulldown(0, FIXEDX, seloption(f), f->pulldown, PACKS);
        f->p->fixedsize.x=f->pulldown->r.max.x-f->pulldown->r.min.x;
        break;
    case TEXTWIN:
        f->p=plframe(0,0);
        pllabel(f->p, PACKN|FILLX, f->name);
        scrl=plscrollbar(f->p, PACKW|FILLY);
        f->textwin=pledit(f->p, EXPAND, Pt(f->cols*chrwidth, f->rows*font->height), 0, 0, 0);
        f->textwin->userp=f;
        plscroll(f->textwin, 0, scrl);
        break;
    case INDEX:
        f->p=plentry(0, 0, f->size*chrwidth, f->value, h_submitindex);
        break;
    }
    if(f->p){
        f->p->userp=f;
        free(t->text);
        t->text=0;
        t->p=f->p;
        t->flags|=PL_HOT;
    }
}
```

⟨*function* h_checkinput*(mothra)* 147a⟩≡ (153b)

```
void h_checkinput(Panel *p, int, int v){
    ((Field *)p->userp)->state=v;
}
```

⟨*function* h_radioinput*(mothra)* 147b⟩≡ (153b)

```
void h_radioinput(Panel *p, int, int v){
    Field *f, *me;
    me=p->userp;
    me->state=v;
    if(v){
        for(f=me->form->fields;f;f=f->next)
            if(f->type==RADIO && f!=me && strcmp(f->name, me->name)==0){
                plsetbutton(f->p, 0);
                f->state=0;
                pldraw(f->p, view);
            }
    }
}
```

⟨*function* h_select*(mothra)* 147c⟩≡ (153b)

```
void h_select(Panel *p, int, int index){
    Option *a;
    Field *f;
    f=p->userp;
    if(f==0) return;
    if(!f->multiple) for(a=f->options;a;a=a->next) a->selected=0;
    for(a=f->options;index!=0 && a!=0;--index,a=a->next);
    if(a==0) return;
    a->selected=!a->selected;
    plinitpulldown(f->p, FIXEDX, seloption(f), f->pulldown, PACKS);
    pldraw(f->p, view);
}
```

⟨*function* h_resetinput*(mothra)* 148a⟩≡ (153b)

```
void h_resetinput(Panel *p, int){
    Field *f;
    Option *o;
    for(f=((Field *)p->userp)->form->fields;f;f=f->next) switch(f->type){
    case TYPEIN:
        plinitentry(f->p, 0, f->size*chrwidth, f->value, 0);
        break;
    case PASSWD:
        plinitentry(f->p, USERFL, f->size*chrwidth, f->value, 0);
        break;
    case FILE:
        free(f->value);
        f->value=strdup("");
        if(f->p==nil) break;
        f->p->state=0;
        pldraw(f->p, view);
        break;
    case CHECK:
    case RADIO:
        f->state=f->checked;
        plsetbutton(f->p, f->checked);
        break;
    case SELECT:
        for(o=f->options;o;o=o->next)
            o->selected=o->def;
        break;
    }
    pldraw(text, view);
}
```

⟨*function* h_buttoninput*(mothra)* 148b⟩≡ (153b)

```
void h_buttoninput(Panel *p, int){
    Field *f;

    f = p->userp;
    if(f && f->form && f->form->method != POST && f->form->action)
        geturl(f->form->action, -1, 0, 0);
}
```

⟨*function* h_fileinput*(mothra)* 148c⟩≡ (153b)

```
void h_fileinput(Panel *p, int){
    char name[NNAME];
    Field *f;

    f = p->userp;
    nstrcpy(name, f->value, sizeof(name));
    for(;;){
        if(eenter("Upload file", name, sizeof(name), &mouse) <= 0)
            break;
        if(access(name, AREAD) == 0)
            break;
    }
    free(f->value);
    f->value = strdup(name);
    p->state = name[0] != 0;
    pldraw(f->p, view);
}
```

⟨*function* h_submittype*(mothra)* 149a⟩≡ (153b)

```
/*
 * If there's exactly one button with type=text, then
 * a CR in the button is supposed to submit the form.
 */
void h_submittype(Panel *p, char *){
    int ntype;
    Field *f;
    ntype=0;
    for(f=((Field *)p->userp)->form->fields;f;f=f->next)
        if(f->type==TYPEIN || f->type==PASSWD)
            ntype++;
    if(ntype==1) h_submitinput(p, 0);
}
```

⟨*function* h_submitindex*(mothra)* 149b⟩≡ (153b)

```
void h_submitindex(Panel *p, char *){
    h_submitinput(p, 0);
}
```

⟨*function* mencodeform*(mothra)* 149c⟩≡ (153b)

```
void mencodeform(Form *form, int fd){
    char *b, *p, *sep;
    int ifd, n, nb;
    Option *o;
    Field *f;
    Rune *rp;

    sep = "--" BOUNDARY;
    for(f=form->fields;f;f=f->next)switch(f->type){
    case TYPEIN:
    case PASSWD:
        fprint(fd, "%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n\r\n%s",
            sep, f->name, plentryval(f->p));
        sep = "\r\n--" BOUNDARY;
        break;
    case CHECK:
    case RADIO:
    case SUBMIT:
        if(!f->state) break;
    case HIDDEN:
        if(f->name==0 || f->value==0)
            break;
        fprint(fd, "%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n\r\n%s",
            sep, f->name, f->value);
        sep = "\r\n--" BOUNDARY;
        break;
    case SELECT:
        if(f->name==0) break;
        for(o=f->options;o;o=o->next)
            if(o->selected && o->value){
                fprint(fd, "%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n\r\n%s",
                    sep, f->name, o->value);
                sep = "\r\n--" BOUNDARY;
            }
        break;
    case TEXTWIN:
        if(f->name==0) break;
        n=plelen(f->textwin);
        rp=pleget(f->textwin);
```

```
        p=b=malloc(UTFmax*n+1);
        if(b == nil)
            break;
        while(n > 0){
            p += runetochar(p, rp);
            rp++;
            n--;
        }
        *p = 0;
        fprint(fd, "%s\r\nContent-Disposition: form-data; name=\"%s\"\r\n\r\n%s",
            sep, f->name, b);
        sep = "\r\n--" BOUNDARY;
        free(b);
        break;
    case FILE:
        if(f->name==0 || f->value[0]==0)
            break;
        if(p = strrchr(f->value, '/'))
            p++;
        if(p == 0 || *p == 0)
            p = f->value;
        if((b = malloc(nb = 8192)) == nil)
            break;
        if((ifd = open(f->value, OREAD)) >= 0){
            if(filetype(ifd, b, nb) < 0)
                strcpy(b, "application/octet-stream");
            fprint(fd, "%s\r\nContent-Disposition: form-data; name=\"%s\"; filename=\"%s\""
                "\r\nContent-Type: %s\r\n\r\n", sep, f->name, p, b);
            sep = "\r\n--" BOUNDARY;
            while((n = read(ifd, b, nb)) > 0)
                if(write(fd, b, n) != n)
                    break;
            close(ifd);
        }
        free(b);
        break;
    }
    fprint(fd, "%s--\r\n", sep);
}
```

⟨*function* uencodeform*(mothra)* 150⟩≡ (153b)

```
void uencodeform(Form *form, int fd){
    char *b, *p, *sep;
    Option *o;
    Field *f;
    Rune *rp;
    int n;

    sep = "";
    for(f=form->fields;f;f=f->next) switch(f->type){
    case TYPEIN:
    case PASSWD:
        fprint(fd, "%s%U=%U", sep, f->name, plentryval(f->p));
        sep = "&";
        break;
    case INDEX:
        fprint(fd, "%s%U", sep, plentryval(f->p));
        sep = "&";
        break;
    case CHECK:
```

```
    case RADIO:
    case SUBMIT:
        if(!f->state) break;
    case HIDDEN:
        if(f->name==0 || f->value==0)
            break;
        fprint(fd, "%s%U=%U", sep, f->name, f->value);
        sep = "&";
        break;
    case SELECT:
        if(f->name==0) break;
        for(o=f->options;o;o=o->next)
            if(o->selected && o->value){
                fprint(fd, "%s%U=%U", sep, f->name, o->value);
                sep = "&";
            }
        break;
    case TEXTWIN:
        if(f->name==0) break;
        n=plelen(f->textwin);
        rp=pleget(f->textwin);
        p=b=malloc(UTFmax*n+1);
        if(b == nil)
            break;
        while(n > 0){
            p += runetochar(p, rp);
            rp++;
            n--;
        }
        *p = 0;
        fprint(fd, "%s%U=%U", sep, f->name, b);
        sep = "&";
        free(b);
        break;
    }
}
```

⟨*function* h_submitinput*(mothra)* 151⟩≡ (153b)

```
void h_submitinput(Panel *p, int){
    char buf[NNAME];
    Form *form;
    Field *f;
    int n, fd;

    f = p->userp;
    form=f->form;
    for(f=form->fields;f;f=f->next)
        if(f->type==SUBMIT)
            f->state = (f->p == p);

    switch(form->method){
    case GET:
        strcpy(buf, "/tmp/mfXXXXXXXXXXX");
        fd = create(mktemp(buf), ORDWR|ORCLOSE, 0600);
        break;
    case POST:
        fd = urlpost(selurl(form->action), form->ctype);
        break;
    default:
        return;
```

```
    }

    if(fd < 0){
        message("submit: %r");
        return;
    }
    if(form->method==GET){
        fprint(fd, "%s?", form->action);
        uencodeform(form, fd);
        seek(fd, 0, 0);
        n = readn(fd, buf, sizeof(buf));
        close(fd);
        if(n < 0 || n >= sizeof(buf)){
            message("submit: too large");
            return;
        }
        buf[n] = 0;
        geturl(buf, -1, 0, 0);
    } else {
        /* only set for multipart/form-data */
        if(form->ctype)
            mencodeform(form, fd);
        else
            uencodeform(form, fd);
        geturl(form->action, fd, 0, 0);
    }
}
```

⟨*function* `freeform`*(mothra)* 152⟩≡ (153b)

```
void freeform(void *p)
{
    Form *form;
    Field *f;
    Option *o;

    while(form = p){
        p = form->next;
        free(form->action);
        while(f = form->fields){
            form->fields = f->next;

            if(f->p!=0)
                plfree(f->p);

            free(f->name);
            free(f->value);

            while(o = f->options){
                f->options = o->next;
                if(o->value != o->label+1)
                    free(o->value);
                free(o);
            }

            free(f);
        }
        free(form);
    }
}
```

⟨*function* `Ufmt` *(mothra)* 153a⟩≡ (153b)

```
int Ufmt(Fmt *f){
    char *s = va_arg(f->args, char*);
    for(; *s; s++){
        if(strchr("/$-_@.!*'(),", *s)
        || 'a'<=*s && *s<='z'
        || 'A'<=*s && *s<='Z'
        || '0'<=*s && *s<='9')
            fmtprint(f, "%c", *s);
        else if(*s==' ')
            fmtprint(f, "+");
        else
            fmtprint(f, "%%%.2X", *s & 0xFF);
    }
    return 0;
}
```

⟨`mothra/forms.c` 153b⟩≡

```
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <event.h>
#include <panel.h>
#include "rtext.h"
#include "mothra.h"
#include "html.h"

typedef struct Field Field;
typedef struct Option Option;
```

⟨*struct* `Form` *(mothra)* 140a⟩

⟨*struct* `Field` *(mothra)* 140b⟩

⟨*enum* `FieldType` *(mothra)* 140c⟩

⟨*struct* `Option` *(mothra)* 141a⟩

⟨*constant* `BOUNDARY` *(mothra)* 141b⟩

```
void h_checkinput(Panel *, int, int);
void h_radioinput(Panel *, int, int);
void h_submitinput(Panel *, int);
void h_buttoninput(Panel *, int);
void h_fileinput(Panel *, int);
void h_submittype(Panel *, char *);
void h_submitindex(Panel *, char *);
void h_resetinput(Panel *, int);
void h_select(Panel *, int, int);

char *selgen(Panel *, int);
char *nullgen(Panel *, int);
```

⟨*function* `newfield` *(mothra)* 141c⟩
⟨*function* `rdform` *(mothra)* 141d⟩
⟨*function* `endform` *(mothra)* 145a⟩
⟨*function* `nullgen` *(mothra)* 145b⟩
⟨*function* `selgen` *(mothra)* 145c⟩
⟨*function* `seloption` *(mothra)* 146a⟩
⟨*function* `mkfieldpanel` *(mothra)* 146b⟩

⟨*function* `h_checkinput` *(mothra)* 147a⟩
⟨*function* `h_radioinput` *(mothra)* 147b⟩
⟨*function* `h_select` *(mothra)* 147c⟩
⟨*function* `h_resetinput` *(mothra)* 148a⟩

⟨*function* `h_buttoninput` *(mothra)* 148b⟩

⟨*function* `h_fileinput` *(mothra)* 148c⟩

⟨*function* `h_submittype` *(mothra)* 149a⟩
⟨*function* `h_submitindex` *(mothra)* 149b⟩

⟨*function* `mencodeform` *(mothra)* 149c⟩

⟨*function* `uencodeform` *(mothra)* 150⟩

⟨*function* `h_submitinput` *(mothra)* 151⟩

⟨*function* `freeform` *(mothra)* 152⟩

⟨*function* `Ufmt` *(mothra)* 153a⟩

mothra/getpix.c

⟨*struct* `Pix` *(mothra)* 154a⟩≡ (157c)

```
struct Pix{
    Pix *next;
    Image *b;
    int width;
    int height;
    char name[NNAME];
};
```

⟨*global* `pixcmd` *(mothra)* 154b⟩≡ (157c)

```
char *pixcmd[]={
[GIF]   "gif -9t",
[JPEG]  "jpg -9t",
[PNG]   "png -9t",
[BMP]   "bmp -9t",
[ICO]   "ico -c",
};
```

⟨*function* `getimage` *(mothra)* 154c⟩≡ (157c)

```
void getimage(Rtext *t, Www *w){
    Action *ap;
    Url *url;
    Image *b;
    int fd, typ;
    char err[512], buf[80], *s;
    Pix *p;

    ap=t->user;
    url=emalloc(sizeof(Url));
    seturl(url, ap->image, w->url->fullname);
    if(debug)
        fprint(STDERR, "getimage: %s from %s\n", ap->image, w->url->fullname);

    if(urlresolve(url) < 0) {
        sysfatal("xxx: %r");
```

```
};
//snprint(url->fullname, sizeof(url->fullname), "%s%s",
//        w->url->fullname,
//        ap->image
//        );
//url->reltext[0] = 0;
//url->basename[0] = 0;

if(debug)
    fprint(STDERR, "getimage resolved: %s\n", urlstr(url));

for(p=w->pix;p!=nil; p=p->next)
    if(strcmp(ap->image, p->name)==0 && ap->width==p->width && ap->height==p->height){
        t->b = p->b;
        w->changed=true;
        return;
    }
fd=urlget(url, -1);
if(fd==-1){
Err:
    snprint(err, sizeof(err), "[img: %s: %r]", urlstr(url));
    free(t->text);
    t->text=strdup(err);
    w->changed=true;
    close(fd);
    goto Out;
}
typ = snooptype(fd);
if(typ < 0 || typ >= nelem(pixcmd) || pixcmd[typ] == nil){
    werrstr("unknown image type");
    goto Err;
}
if((fd = pipeline(fd, "exec %s", pixcmd[typ])) < 0)
    goto Err;
if(ap->width>0 || ap->height>0){
    s = buf;
    /*
     * claude: under -d, save a copy of each decoded bitmap that's
     * about to be piped into resize, one file per image. If resize
     * crashes on an image, the corresponding .bit file survives and
     * can be replayed as `resize -x N -y N < /tmp/mothra-NNN.bit`.
     * We use a counter rather than a hash of the image URL because
     * mothra keeps loading images after a child resize crashes, and
     * we want every intermediate bitmap preserved.
     */
    static int imgcount;
    if(debug)
        s += sprint(s, "tee /tmp/mothra-%d.bit | ", imgcount);
    s += sprint(s, "exec resize");
    if(ap->width>0)
        s += sprint(s, " -x %d", ap->width);
    if(ap->height>0)
        s += sprint(s, " -y %d", ap->height);
    USED(s);
    if(debug){
        fprint(STDERR, "getimage[%d]: piping %s through `%s` for %s\n",
            imgcount, pixcmd[typ], buf, ap->image);
        imgcount++;
    }
    if((fd = pipeline(fd, buf)) < 0)
```

```
            goto Err;
        }
        b=readimage(display, fd, 1);
        if(b==0){
            werrstr("can't read image");
            goto Err;
        }
        close(fd);
        p=emalloc(sizeof(Pix));
        nstrcpy(p->name, ap->image, sizeof(p->name));
        p->b=b;
        p->width=ap->width;
        p->height=ap->height;
        p->next=w->pix;
        w->pix=p;
        t->b=b;
        w->changed=true;
    Out:
        freeurl(url);
    }
```

⟨*function* `getpix` *(mothra)* 156⟩≡ (157c)

```
    void getpix(Rtext *t, Www *w){
        int i, pid, nworker, worker[NXPROC];
        Action *ap;

        nworker = 0;
        for(i=0; i<nelem(worker); i++)
            worker[i] = -1;

        for(;t!=0;t=t->next){
            ap=t->user;
            if(ap && ap->image){
                pid = rfork(RFFDG|RFPROC|RFMEM);
                switch(pid){
                case -1:
                    fprint(2, "fork: %r\n");
                    break;
                case 0:
                    getimage(t, w);
                    exits(0);
                default:
                    for(i=0; i<nelem(worker); i++)
                        if(worker[i] == -1){
                            worker[i] = pid;
                            nworker++;
                            break;
                        }

                    while(nworker == nelem(worker)){
                        if((pid = waitpid()) < 0)
                            break;
                        for(i=0; i<nelem(worker); i++)
                            if(worker[i] == pid){
                                worker[i] = -1;
                                nworker--;
                                break;
                            }
                    }
                }
```

```
        }
    }
    while(nworker > 0){
        if((pid = waitpid()) < 0)
            break;
        for(i=0; i<nelem(worker); i++)
            if(worker[i] == pid){
                worker[i] = -1;
                nworker--;
                break;
            }
    }
}
```

⟨*function* `countpix` *(mothra)* 157a⟩≡ (157c)

```
ulong countpix(void *p){
    ulong n=0;
    Pix *x;
    for(x = p; x; x = x->next)
        n += Dy(x->b->r)*bytesperline(x->b->r, x->b->depth);
    return n;
}
```

⟨*function* `freepix` *(mothra)* 157b⟩≡ (157c)

```
void freepix(void *p){
    Pix *x, *xx;
    xx = p;
    while(x = xx){
        xx = x->next;
        freeimage(x->b);
        free(x);
    }
}
```

⟨`mothra/getpix.c` 157c⟩≡

```
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <event.h>
#include <panel.h>
#include "mothra.h"

typedef struct Pix Pix;
```

⟨*struct* `Pix` *(mothra)* 154a⟩

⟨*global* `pixcmd` *(mothra)* 154b⟩

⟨*function* `getimage` *(mothra)* 154c⟩

⟨*function* `getpix` *(mothra)* 156⟩

⟨*function* `countpix` *(mothra)* 157a⟩

⟨*function* `freepix` *(mothra)* 157b⟩

mothra/html.syntax.c

⟨*global* tag*(mothra)* 158⟩≡ (159a)

```
Tag tag[]={
[Tag_a]          "a",           END,
[Tag_abbr]       "abbr",        END,
[Tag_acronym]    "acronym",     END,
[Tag_address]    "address",     END,
[Tag_applet]     "applet",      NOEND,
[Tag_article]    "article",     END,
[Tag_audio]      "audio",       OPTEND,
[Tag_b]          "b",           END,
[Tag_base]       "base",        NOEND,
[Tag_blockquot] "blockquote",   END,
[Tag_body]       "body",        END,    /* OPTEND */
[Tag_br]         "br",          NOEND,
[Tag_button]     "button",      END,
[Tag_center]     "center",      END,
[Tag_cite]       "cite",        END,
[Tag_code]       "code",        END,
[Tag_comment]    "!--",         NOEND,
[Tag_dd]         "dd",          NOEND,  /* OPTEND */
[Tag_del]        "del",         END,
[Tag_dfn]        "dfn",         END,
[Tag_dir]        "dir",         END,
[Tag_div]        "div",         END,    /* OPTEND */
[Tag_dl]         "dl",          END,
[Tag_dt]         "dt",          NOEND,  /* OPTEND */
[Tag_em]         "em",          END,
[Tag_embed]      "embed",       NOEND,
[Tag_end]        0,             ERR,
[Tag_figure]     "figure",      END,
[Tag_figcaption]        "figcaption",   NOEND,
[Tag_font]       "font",        END,
[Tag_form]       "form",        END,
[Tag_frame]      "frame",       NOEND,
[Tag_h1]         "h1",          END,
[Tag_h2]         "h2",          END,
[Tag_h3]         "h3",          END,
[Tag_h4]         "h4",          END,
[Tag_h5]         "h5",          END,
[Tag_h6]         "h6",          END,
[Tag_head]       "head",        END,    /* OPTEND */
[Tag_hr]         "hr",          NOEND,
[Tag_html]       "html",        END,    /* OPTEND */
[Tag_i]          "i",           END,
[Tag_iframe]     "iframe",      NOEND,
[Tag_img]        "img",         NOEND,
[Tag_image]      "image",       NOEND,
[Tag_input]      "input",       NOEND,
[Tag_ins]        "ins",         END,
[Tag_isindex]    "isindex",     NOEND,
[Tag_kbd]        "kbd",         END,
[Tag_key]        "key",         END,
[Tag_li]         "li",          NOEND,  /* OPTEND */
[Tag_link]       "link",        NOEND,
[Tag_listing]    "listing",     END,
[Tag_menu]       "menu",        END,
[Tag_meta]       "meta",        NOEND,
[Tag_nextid]     "nextid",      NOEND,
```

```
  [Tag_object]    "object",       END,
  [Tag_ol]        "ol",           END,
  [Tag_option]    "option",       NOEND,  /* OPTEND */
  [Tag_p]         "p",            NOEND,  /* OPTEND */
  [Tag_plaintext] "plaintext",    NOEND,
  [Tag_pre]       "pre",          END,
  [Tag_s]         "s",            END,
  [Tag_samp]      "samp",         END,
  [Tag_script]    "script",       END,
  [Tag_select]    "select",       END,
  [Tag_span]      "span",         END,
  [Tag_strike]    "strike",       END,
  [Tag_strong]    "strong",       END,
  [Tag_style]     "style",        END,
  [Tag_sub]       "sub",          END,
  [Tag_sup]       "sup",          END,
  [Tag_source]    "source",       NOEND,
  [Tag_table]     "table",        END,
  [Tag_td]        "td",           END,
  [Tag_th]        "th",           END,
  [Tag_textarea]  "textarea",     END,
  [Tag_title]     "title",        END,
  [Tag_tr]        "tr",           END,
  [Tag_tt]        "tt",           END,
  [Tag_u]         "u",            END,
  [Tag_ul]        "ul",           END,
  [Tag_var]       "var",          END,
  [Tag_video]     "video",        OPTEND,
  [Tag_wbr]       "wbr",          NOEND,
  [Tag_xmp]       "xmp",          END,
  };
```

⟨mothra/html.syntax.c 159a⟩≡

```
  #include <u.h>
  #include <libc.h>
  #include <draw.h>
  #include <event.h>
  #include <panel.h>
  #include "mothra.h"
  #include "html.h"

  ⟨global tag(mothra) 158⟩
```

mothra/mothra.c

⟨global root(mothra) 159b⟩≡ (181)

```
  Panel *root;    /* the whole display */
```

⟨global alt(mothra) 159c⟩≡ (181)

```
  Panel *alt;     /* the alternate display */
```

⟨global alttext(mothra) 159d⟩≡ (181)

```
  Panel *alttext; /* the alternate text window */
```

⟨global cmd(mothra) 159e⟩≡ (181)

```
  Panel *cmd;     /* command entry */
```

⟨global cururl(mothra) 159f⟩≡ (181)

```
  Panel *cururl;  /* label giving the url of the visible text */
```

⟨*global* `list` *(mothra)* 160a⟩≡ (181)
```
Panel *list;    /* list of previously acquired www pages */
```

⟨*global* `msg` *(mothra)* 160b⟩≡ (181)
```
Panel *msg;     /* message display */
```

⟨*global* `menu3` *(mothra)* 160c⟩≡ (181)
```
Panel *menu3;   /* button 3 menu */
```

⟨*global* `mothra` *(mothra)* 160d⟩≡ (181)
```
char mothra[] = "mothra!";
```

⟨*global* `current` *(mothra)* 160e⟩≡ (181)
```
Www *current=nil;
```

⟨*global* `selection` *(mothra)* 160f⟩≡ (181)
```
Url *selection=nil;
```

⟨*function* `www` *(mothra)* 160g⟩≡ (181)
```
Www *www(int index){
    static Www a[NWWW];
    return &a[index % NWWW];
}
```

⟨*function* `nwww` *(mothra)* 160h⟩≡ (181)
```
int nwww(void){
    return wwwtop<NWWW ? wwwtop : NWWW;
}
```

⟨*function* `subpanel` *(mothra)* 160i⟩≡ (181)
```
int subpanel(Panel *obj, Panel *subj){
    if(obj==0) return 0;
    if(obj==subj) return 1;
    for(obj=obj->child;obj;obj=obj->next)
        if(subpanel(obj, subj)) return 1;
    return 0;
}
```

⟨*function* `adjkb` *(mothra)* 160j⟩≡ (181)
```
/*
 * Make sure that the keyboard focus is on-screen, by adjusting it to
 * be the cmd entry if necessary.
 */
int adjkb(void){
    Rtext *t;
    int yoffs;
    if(current){
        yoffs=text->r.min.y-plgetpostextview(text);
        for(t=current->text;t;t=t->next) if(!eqrect(t->r, Rect(0,0,0,0))){
            if(t->r.max.y+yoffs>=text->r.min.y
            && t->r.min.y+yoffs<text->r.max.y
            && t->b==0
            && subpanel(t->p, plkbfocus))
                return 1;
        }
    }
    plgrabkb(cmd);
    return 0;
}
```

⟨*function* `scrollpanel` *(mothra)* 161a⟩≡ (181)

```
void scrollpanel(Panel *p, int dy, int whence)
{
    Scroll s;

    s = plgetscroll(p);
    switch(whence){
    case 0:
        s.pos.y = dy;
        break;
    case 1:
        s.pos.y += dy;
        break;
    case 2:
        s.pos.y = s.size.y+dy;
        break;
    }
    if(s.pos.y > s.size.y)
        s.pos.y = s.size.y;
    if(s.pos.y < 0)
        s.pos.y = 0;
    plsetscroll(p, s);
}
```

⟨*function* `sidescroll` *(mothra)* 161b⟩≡ (181)

```
void sidescroll(int dx, int whence)
{
    Scroll s;

    s = plgetscroll(text);
    switch(whence){
    case 0:
        s.pos.x = dx;
        break;
    case 1:
        s.pos.x += dx;
        break;
    case 2:
        s.pos.x = s.size.x+dx;
        break;
    }
    if(s.pos.x > s.size.x - text->size.x + 5)
        s.pos.x = s.size.x - text->size.x + 5;
    if(s.pos.x < 0)
        s.pos.x = 0;
    plsetscroll(text, s);
}
```

⟨*function* `mkpanels` *(mothra)* 161c⟩≡ (181)

```
void mkpanels(void){
    Panel *p, *xbar, *ybar, *swap;
    int xflags;

    if(topxbar)
        xflags=PACKN|USERFL;
    else
        xflags=PACKS|USERFL;
    //TODO
    //if(!visxbar)
    //  xflags|=IGNORE;
```

```
    menu3=plmenu(0, 0, buttons_, PACKN|FILLX, hit3);
    root=plpopup(root, EXPAND, 0, 0, menu3);
        p=plgroup(root, PACKN|FILLX);
            msg=pllabel(p, PACKN|FILLX, mothra);
            plplacelabel(msg, PLACEW);
            pllabel(p, PACKW, "Go:");
            cmd=plentry(p, PACKN|FILLX, 0, "", docmd);
        p=plgroup(root, PACKN|FILLX);
            ybar=plscrollbar(p, PACKW);
            list=pllist(p, PACKN|FILLX, genwww, 8, doprev);
            plscroll(list, 0, ybar);
        p=plgroup(root, PACKN|FILLX);
            pllabel(p, PACKW, "Url:");
            cururl=pllabel(p, PACKE|EXPAND, "---");
            plplacelabel(cururl, PLACEW);
        p=plgroup(root, PACKN|EXPAND);
            ybar=plscrollbar(p, PACKW|USERFL);
            xbar=plscrollbar(p, xflags);
            text=pltextview(p, PACKE|EXPAND, Pt(0, 0), 0, dolink);
            plscroll(text, xbar, ybar);
    plgrabkb(cmd);
    alt=plpopup(0, PACKE|EXPAND, 0, 0, menu3);
        ybar=plscrollbar(alt, PACKW|USERFL);
        xbar=plscrollbar(alt, xflags);
        alttext=pltextview(alt, PACKE|EXPAND, Pt(0, 0), 0, dolink);
        plscroll(alttext, xbar, ybar);
    if(!defdisplay){
        swap=root;
        root=alt;
        alt=swap;
        swap=text;
        text=alttext;
        alttext=swap;
    }
}
```

⟨*function* killcohort*(mothra)* 162a⟩≡ (181)

```
void killcohort(void){
    int i;
    for(i=0;i!=3;i++){  /* It's a long way to the kitchen */
        postnote(PNGROUP, cohort, "kill\n");
        sleep(1);
    }
}
```

⟨*function* catch*(mothra)* 162b⟩≡ (181)

```
void catch(void*, char*){
    noted(NCONT);
}
```

⟨*function* dienow*(mothra)* 162c⟩≡ (181)

```
void dienow(void*, char*){
    noted(NDFLT);
}
```

⟨*function* mkhome*(mothra)* 162d⟩≡ (181)

```
char* mkhome(void){
    static char *home;          /* where to put files */
    char *henv, *tmp;
    int f;
```

```
    if(home == nil){
        henv=getenv("home");
        if(henv){
            tmp = smprint("%s/lib", henv);
            f=create(tmp, OREAD, DMDIR|0777);
            if(f!=-1) close(f);
            free(tmp);

            home = smprint("%s/lib/mothra", henv);
            f=create(home, OREAD, DMDIR|0777);
            if(f!=-1) close(f);
            free(henv);
        }
        else
            home = strdup("/tmp");
    }
    return home;
}
```

⟨*function* `donecurs` *(mothra)* 163a⟩≡ (181)

```
void donecurs(void){
    if(current && current->alldone==0)
        esetcursor(&readingcurs);
    else if(mothmode)
        esetcursor(&mothcurs);
    else
        esetcursor(0);
}
```

⟨*function* `drawlock` *(mothra)* 163b⟩≡ (181)

```
void drawlock(int dolock){
    static int ref = 0;
    if(dolock){
        if(ref++ == 0)
            lockdisplay(display);
    } else {
        if(--ref == 0)
            unlockdisplay(display);
    }
}
```

⟨*function* `main` *(mothra)* 163c⟩≡ (181)

```
void main(int argc, char *argv[]){
    Event e;
    enum { Eplumb = 128, Ekick = 256 };
    Plumbmsg *pm;
    char *url;
    int i;

    quotefmtinstall();
    fmtinstall('U', Ufmt);

    ARGBEGIN{
    case 'd': debug=1; break;
    case 'v': verbose=1; break;
    case 'k': killimgs=1; break;
    case 'm':
        if(mtpt = ARGF())
            break;
```

```
case 'a': defdisplay=0; break;
default:  goto Usage;
}ARGEND


/*
 * so that we can stop all subprocesses with a note,
 * and to isolate rendezvous from other processes
 */
if(cohort=rfork(RFPROC|RFNOTEG|RFNAMEG|RFREND)){
    atexit(killcohort);
    notify(catch);
    waitpid();
    exits(0);
}
cohort = getpid();
atexit(killcohort);

switch(argc){
default:
Usage:
    fprint(2, "usage: %s [-dvak] [-m mtpt] [url]\n", argv0);
    exits("usage");
case 0:
    url=getenv("url");
    break;
case 1: url=argv[0]; break;
}
if(initdraw(0, 0, mothra) < 0)
    sysfatal("initdraw: %r");
display->locking = 1;
chrwidth=stringwidth(font, "0");
pltabsize(chrwidth, 8*chrwidth);
einit(Emouse|Ekeyboard);
eplumb(Eplumb, "web");
if(pipe(kickpipe) < 0)
    sysfatal("pipe: %r");
estart(Ekick, kickpipe[0], 256);
plinit(view->depth);
if(debug) notify(dienow);
getfonts();
hrule=allocimage(display, Rect(0, 0, 1, 5), view->chan, 1, DWhite);
if(hrule==0)
    sysfatal("can't allocimage!");
draw(hrule, Rect(0,1,1,3), display->black, 0, ZP);
linespace=allocimage(display, Rect(0, 0, 1, 5), view->chan, 1, DWhite);
if(linespace==0)
    sysfatal("can't allocimage!");
bullet=allocimage(display, Rect(0,0,25, 8), view->chan, 0, DWhite);
fillellipse(bullet, Pt(4,4), 3, 3, display->black, ZP);
mkpanels();
unlockdisplay(display);
eresized(0);
drawlock(1);

if(url && url[0])
    geturl(url, -1, 1, 0);

mouse.buttons=0;
for(;;){
```

```
        if(mouse.buttons==0 && current){
            if(current->finished){
                updtext(current);
                if(current->url->tag[0])
                    scrollto(current->url->tag);
                current->finished=0;
                current->changed=false;
                current->alldone=1;
                message(mothra);
                donecurs();
            }
        }

        drawlock(0);
        i=event(&e);
        drawlock(1);

        switch(i){
        case Ekick:
            if(mouse.buttons==0 && current && current->changed){
                if(!current->finished)
                    updtext(current);
                current->changed=false;
            }
            break;
        case Ekeyboard:
            switch(e.kbdc){
            default:
Plkey:
                adjkb();
                plkeyboard(e.kbdc);
                break;
            case Khome:
                scrollpanel(text, 0, 0);
                break;
            case Kup:
                scrollpanel(text, -text->size.y/4, 1);
                break;
            case Kpgup:
                scrollpanel(text, -text->size.y/2, 1);
                break;
            case Kdown:
                scrollpanel(text, text->size.y/4, 1);
                break;
            case Kpgdown:
                scrollpanel(text, text->size.y/2, 1);
                break;
            case Kend:
                scrollpanel(text, -text->size.y, 2);
                break;
            case Kack:
                search();
                break;
            case Kright:
                if(plkbfocus)
                    goto Plkey;
                sidescroll(text->size.x/4, 1);
                break;
            case Kleft:
                if(plkbfocus)
```

```
                goto Plkey;
            sidescroll(-text->size.x/4, 1);
            break;
        }
        break;
    case Emouse:
        mouse=e.mouse;
        if(mouse.buttons & (8|16) && ptinrect(mouse.xy, list->r) && defdisplay){
            if(mouse.buttons & 8)
                scrollpanel(list, list->r.min.y - mouse.xy.y, 1);
            else
                scrollpanel(list, mouse.xy.y - list->r.min.y, 1);
            break;
        }
        if(mouse.buttons & (8|16) && ptinrect(mouse.xy, text->r)){
            if(mouse.buttons & 8)
                scrollpanel(text, text->r.min.y - mouse.xy.y, 1);
            else
                scrollpanel(text, mouse.xy.y - text->r.min.y, 1);
            break;
        }
        plmouse(root, &mouse);
        if(mouse.buttons == 1 && root->lastmouse == root)
            plgrabkb(nil);
        break;
    case Eplumb:
        pm=e.v;
        if(pm->ndata > 0)
            geturl(pm->data, -1, 1, 0);
        plumbfree(pm);
        break;
    }
  }
```

⟨*function* confirm*(mothra)* 166a⟩≡ (181)

```
  int confirm(int b){
      Mouse down, up;
      esetcursor(&confirmcursor);
      do down=emouse(); while(!down.buttons);
      do up=emouse(); while(up.buttons);
      donecurs();
      return down.buttons==(1<<(b-1));
  }
```

⟨*function* message*(mothra)* 166b⟩≡ (181)

```
  void message(char *s, ...){
      static char buf[1024];
      char *out;
      va_list args;
      va_start(args, s);
      out = buf + vsnprint(buf, sizeof(buf), s, args);
      va_end(args);
      *out='\0';
      plinitlabel(msg, PACKN|FILLX, buf);
      if(defdisplay) pldraw(msg, view);
  }
```

⟨*function* htmlerror*(mothra)* 166c⟩≡ (181)

```
  void htmlerror(char *name, int line, char *m, ...){
```

```
    static char buf[1024];
    char *out;
    va_list args;
    if(verbose){
        va_start(args, m);
        out=buf+snprint(buf, sizeof(buf), "%s: line %d: ", name, line);
        out+=vsnprint(out, sizeof(buf)-(out-buf)-1, m, args);
        va_end(args);
        *out='\0';
        fprint(2, "%s\n", buf);
    }
}
```

⟨*function* eresized*(mothra)* 167a⟩≡ (181)

```
void eresized(int new){
    Rectangle r;

    drawlock(1);
    if(new && getwindow(display, Refnone) == -1) {
        fprint(2, "getwindow: %r\n");
        exits("getwindow");
    }
    r=view->r;
    plpack(root, r);
    plpack(alt, r);
    pldraw(cmd, view);  /* put cmd box on screen for alt display */
    pldraw(root, view);
    flushimage(display, 1);
    drawlock(0);
}
```

⟨*function* emalloc*(mothra)* 167b⟩≡ (181)

```
void *emalloc(int n){
    void *v;
    v=malloc(n);
    if(v==0)
        sysfatal("out of memory");
    memset(v, 0, n);
    setmalloctag(v, getcallerpc(&n));
    return v;
}
```

⟨*function* nstrcpy*(mothra)* 167c⟩≡ (181)

```
void nstrcpy(char *to, char *from, int len){
    strncpy(to, from, len);
    to[len-1] = 0;
}
```

⟨*function* genwww*(mothra)* 167d⟩≡ (181)

```
char *genwww(Panel *, int index){
    static char buf[1024];
    Www *w;
    int i;

    if(index >= nwww())
        return 0;
    i = wwwtop-index-1;
    w = www(i);
    if(!w->url)
        return 0;
```

```
    if(w->title[0]!='\0'){
        w->gottitle=1;
        snprint(buf, sizeof(buf), "%2d %s", i+1, w->title);
    } else
        snprint(buf, sizeof(buf), "%2d %s", i+1, urlstr(w->url));
    return buf;
}
```

⟨*function* scrollto*(mothra)* 168a⟩≡ (181)

```
void scrollto(char *tag){
    Rtext *tp;
    Action *ap;
    if(current == nil || text == nil)
        return;
    if(tag && tag[0]){
        for(tp=current->text;tp;tp=tp->next){
            ap=tp->user;
            if(ap && ap->name && strcmp(ap->name, tag)==0){
                current->yoffs=tp->topy;
                break;
            }
        }
    }
    plsetpostextview(text, current->yoffs);
}
```

⟨*function* setcurrent*(mothra)* 168b⟩≡ (181)

```
/*
 * selected text should be a url.
 */
void setcurrent(int index, char *tag){
    Www *new;
    int i;
    new=www(index);
    if(new==current && (tag==0 || tag[0]==0)) return;
    if(current)
        current->yoffs=plgetpostextview(text);
    current=new;
    plinitlabel(cururl, PACKE|EXPAND, current->url->fullname);
    if(defdisplay) pldraw(cururl, view);
    plinittextview(text, PACKE|EXPAND, Pt(0, 0), current->text, dolink);
    scrollto(tag);
    if((i = open("/dev/label", OWRITE)) >= 0){
        fprint(i, "%s %s", mothra, current->url->fullname);
        close(i);
    }
    donecurs();
}
```

⟨*function* arg*(mothra)* 168c⟩≡ (181)

```
char *arg(char *s){
    do ++s; while(*s==' ' || *s=='\t');
    return s;
}
```

⟨*function* save*(mothra)* 168d⟩≡ (181)

```
void save(int ifd, char *name){
    char buf[NNAME+64];
    int ofd;
    if(ifd < 0){
```

```
        message("save: %s: %r", name);
        return;
    }
    ofd=create(name, OWRITE, 0666);
    if(ofd < 0){
        message("save: %s: %r", name);
        return;
    }
    switch(rfork(RFNOTEG|RFNAMEG|RFFDG|RFMEM|RFPROC|RFNOWAIT)){
    case -1:
        message("Can't fork: %r");
        break;
    case 0:
        dup(ifd, 0);
        close(ifd);
        dup(ofd, 1);
        close(ofd);

        snprint(buf, sizeof(buf),
            "{tput -p || cat} |[2] {aux/statusmsg -k %q >/dev/null || cat >/dev/null}", name);
        execl("/bin/rc", "rc", "-c", buf, nil);
        exits("exec");
    }
    close(ifd);
    close(ofd);
    donecurs();
}
```

⟨*function* `screendump` *(mothra)* 169a⟩≡ (181)

```
void screendump(char *name, int full){
    Image *b;
    int fd;
    fd=create(name, OWRITE, 0666);
    if(fd==-1){
        message("can't create %s", name);
        return;
    }
    if(full){
        writeimage(fd, view, 0);
    } else {
        if((b=allocimage(display, text->r, view->chan, 0, DNofill)) == nil){
            message("can't allocate image");
            close(fd);
            return;
        }
        draw(b, b->r, view, 0, b->r.min);
        writeimage(fd, b, 0);
        freeimage(b);
    }
    close(fd);
}
```

⟨*function* `urltofile` *(mothra)* 169b⟩≡ (181)

```
/*
 * convert a url into a local file name.
 */
char *urltofile(Url *url){
    char *name, *slash;
    if(url == nil)
        return nil;
```

```
    name = urlstr(url);
    if(name == nil || name[0] == 0)
        name = "/";
    if(slash = strrchr(name, '/'))
        name = slash+1;
    if(name[0] == 0)
        name = "index";
    return name;
}
```

⟨*function* docmd*(mothra)* 170⟩≡ (181)

```
/*
 * user typed a command.
 */
void docmd(Panel *p, char *s){
    char buf[NNAME];
    int c;

    USED(p);
    while(*s==' ' || *s=='\t') s++;
    /*
     * Non-command does a get on the url
     */
    if(s[0]!='\0' && s[1]!='\0' && s[1]!=' ')
        geturl(s, -1, 0, 0);
    else switch(c = s[0]){
    default:
        message("Unknown command %s", s);
        break;
    case 'a':
        s = arg(s);
        if(*s=='\0' && selection)
            hit3(3, 0);
        break;
    case 'd':
        s = arg(s);
        if(*s){
            s = smprint("https://lite.duckduckgo.com/lite/?q=%U&kd=-1", s);
            if(s != nil)
                geturl(s, -1, 0, 0);
            free(s);
        }else
            message("Usage: d text");
        break;
    case 'g':
        s = arg(s);
        if(*s=='\0'){
    case 'r':
            if(selection)
                s = urlstr(selection);
            else
                message("no url selected");
        }
        geturl(s, -1, 0, 0);
        break;
    case 'j':
        s = arg(s);
        if(*s)
            doprev(nil, 1, wwwtop-atoi(s));
        else
```

```
            message("Usage: j index");
        break;
    case 'm':
        mothon(current, !mothmode);
        break;
    case 'k':
        killimgs = !killimgs;
        if (killimgs)
            killpix(current);
        break;
    case 'w':
    case 'W':
        s = arg(s);
        if(s==0 || *s=='\0'){
            snprint(buf, sizeof(buf), "dump.bit");
            if(eenter("Screendump to", buf, sizeof(buf), &mouse) <= 0)
                break;
            s = buf;
        }
        screendump(s, c == 'W');
        break;
    case 's':
        s = arg(s);
        if(!selection){
            message("no url selected");
            break;
        }
        if(s==0 || *s=='\0'){
            snprint(buf, sizeof(buf), "%s", urltofile(selection));
            if(eenter("Save to", buf, sizeof(buf), &mouse) <= 0)
                break;
            s = buf;
        }
        save(urlget(selection, -1), s);
        break;
    case 'q':
        exits(0);
    }
    plinitentry(cmd, EXPAND, 0, "", docmd);
    pldraw(root, view);
}
```

⟨*function* `regerror` *(mothra)* 171a⟩≡ (181)

```
void regerror(char *msg)
{
    werrstr("regerror: %s", msg);
}
```

⟨*function* `search` *(mothra)* 171b⟩≡ (181)

```
void search(void){
    static char last[256];
    char buf[256];
    Reprog *re;
    Rtext *tp;

    for(;;){
        if(current == nil || current->text == nil || text == nil)
            return;
        strncpy(buf, last, sizeof(buf)-1);
        if(eenter("Search for", buf, sizeof(buf), &mouse) <= 0)
```

```
            return;
        strncpy(last, buf, sizeof(buf)-1);
        re = regcompnl(buf);
        if(re == nil){
            message("%r");
            continue;
        }
        for(tp=current->text;tp;tp=tp->next)
            if(tp->flags & PL_SEL)
                break;
        if(tp == nil)
            tp = current->text;
        else {
            tp->flags &= ~PL_SEL;
            tp = tp->next;
        }
        while(tp != nil){
            tp->flags &= ~PL_SEL;
            if(tp->text && *tp->text)
            if(regexec(re, tp->text, nil, 0)){
                tp->flags |= PL_SEL;
                plsetpostextview(text, tp->topy);
                break;
            }
            tp = tp->next;
        }
        free(re);
        updtext(current);
    }
}
```

⟨*function* `hiturl` *(mothra)* 172a⟩≡ (181)

```
void hiturl(int buttons, char *url, int map){
    switch(buttons){
    case 1: geturl(url, -1, 0, map); break;
    case 2: urlresolve(selurl(url)); break;
    case 4: message("Button 3 hit on url can't happen!"); break;
    }
}
```

⟨*function* `doprev` *(mothra)* 172b⟩≡ (181)

```
/*
 * user selected from the list of available pages
 */
void doprev(Panel *p, int buttons, int index){
    int i;
    USED(p);
    if(index < 0 || index >= nwww())
        return;
    i = wwwtop-index-1;
    switch(buttons){
    case 1: setcurrent(i, 0);   /* no break ... */
    case 2: selurl(www(i)->url->fullname); break;
    case 4: message("Button 3 hit on page can't happen!"); break;
    }
}
```

⟨*function* `dolink` *(mothra)* 172c⟩≡ (181)

```
/*
 * Follow an html link
```

```
 */
void dolink(Panel *p, int buttons, Rtext *word){
    Action *a;

    a=word->user;
    if(a == nil || (a->link == nil && a->image == nil))
        return;
    if(mothmode)
        hiturl(buttons, a->image ? a->image : a->link, 0);
    else if(a->link){
        if(a->ismap){
            char mapurl[NNAME];
            Point coord;
            int yoffs;

            yoffs=plgetpostextview(p);
            coord=subpt(subpt(mouse.xy, word->r.min), p->r.min);
            snprint(mapurl, sizeof(mapurl), "%s?%d,%d", a->link, coord.x, coord.y+yoffs);
            hiturl(buttons, mapurl, 1);
        } else
            hiturl(buttons, a->link, 0);
    }
}
```

⟨*function* `filter` *(mothra)* 173a⟩≡ (181)

```
void filter(int fd, char *cmd){
    switch(rfork(RFFDG|RFPROC|RFMEM|RFREND|RFNOWAIT|RFNOTEG)){
    case -1:
        message("Can't fork!");
        break;
    case 0:
        dupfds(fd, 1, 2, -1);
        execl("/bin/rc", "rc", "-c", cmd, nil);
        _exits(0);
    }
    close(fd);
}
```

⟨*function* `gettext` *(mothra)* 173b⟩≡ (181)

```
void gettext(Www *w, int fd, int type){
    switch(rfork(RFFDG|RFPROC|RFMEM|RFNOWAIT)){
    case -1:
        message("Can't fork, please wait");
        break;
    case 0:
        if(type==HTML)
            plrdhtml(w->url->fullname, fd, w, killimgs);
        else
            plrdplain(w->url->fullname, fd, w);
        _exits(0);
    }
    close(fd);
}
```

⟨*function* `freetext` *(mothra)* 173c⟩≡ (181)

```
void freetext(Rtext *t){
    Rtext *tt;
    Action *a;

    tt = t;
```

```
    for(; t!=0; t = t->next){
        t->b=0;
        free(t->text);
        t->text=0;
        if(a = t->user){
            t->user=0;
            free(a->image);
            free(a->link);
            free(a->name);
            free(a);
        }
    }
    plrtfree(tt);
}
```

⟨*function* dupfds*(mothra)* 174a⟩≡ (181)

```
void
dupfds(int fd, ...)
{
    int mfd, n, i;
    va_list arg;
    Dir *dir;

    va_start(arg, fd);
    for(mfd = 0; fd >= 0; fd = va_arg(arg, int), mfd++)
        if(fd != mfd)
            if(dup(fd, mfd) < 0)
                sysfatal("dup: %r");
    va_end(arg);
    if((fd = open("/fd", OREAD)) < 0)
        sysfatal("open: %r");
    n = dirreadall(fd, &dir);
    for(i=0; i<n; i++){
        if(strstr(dir[i].name, "ctl"))
            continue;
        fd = atoi(dir[i].name);
        if(fd >= mfd)
            close(fd);
    }
    free(dir);
}
```

⟨*function* pipeline*(mothra)* 174b⟩≡ (181)

```
int pipeline(int fd, char *fmt, ...)
{
    char buf[80], *argv[4];
    va_list arg;
    int pfd[2];

    va_start(arg, fmt);
    vsnprint(buf, sizeof buf, fmt, arg);
    va_end(arg);

    if(pipe(pfd) < 0){
    Err:
        close(fd);
        werrstr("pipeline for %s failed: %r", buf);
        return -1;
    }
    switch(rfork(RFPROC|RFMEM|RFFDG|RFREND|RFNOWAIT)){
```

```
    case -1:
        close(pfd[0]);
        close(pfd[1]);
        goto Err;
    case 0:
        dupfds(fd, pfd[1], 2, -1);
        argv[0] = "rc";
        argv[1] = "-c";
        argv[2] = buf;
        argv[3] = nil;
        exec("/bin/rc", argv);
        _exits(0);
    }
    close(fd);
    close(pfd[1]);
    return pfd[0];
}
```

⟨*function* `urlstr` *(mothra)* 175a⟩≡ (181)

```
char*
urlstr(Url *url){
    if(url->fullname[0])
        return url->fullname;
    return url->reltext;
}
```

⟨*function* `copyurl` *(mothra)* 175b⟩≡ (181)

```
Url *copyurl(Url *u){
    Url *v;
    v=emalloc(sizeof(Url));
    *v=*u;
    v->reltext = strdup(u->reltext);
    v->basename = strdup(u->basename);
    return v;
}
```

⟨*function* `freeurl` *(mothra)* 175c⟩≡ (181)

```
void freeurl(Url *u){
    free(u->reltext);
    free(u->basename);
    free(u);
}
```

⟨*function* `seturl` *(mothra)* 175d⟩≡ (181)

```
void seturl(Url *url, char *urlname, char *base){
    url->reltext = strdup(urlname);
    url->basename = strdup(base);
    url->fullname[0] = 0;
    url->tag[0] = 0;
    url->map = 0;
}
```

⟨*function* `selurl` *(mothra)* 175e⟩≡ (181)

```
Url* selurl(char *urlname){
    Url *last;

    last=selection;
    selection=emalloc(sizeof(Url));
    seturl(selection, urlname, current ? current->url->fullname : "");
    if(last) freeurl(last);
```

```
    message("selected: %s", urlstr(selection));
    plgrabkb(cmd);              /* for snarf */
    return selection;
}
```

⟨*function* `geturl` *(mothra)* 176⟩≡ (181)

```
/*
 * get the file at the given url
 */
void geturl(char *urlname, int post, int plumb, int map){
    int i, fd, typ;
    char cmd[NNAME];
    ulong n;
    Www *w;

    if(*urlname == '#' && post < 0){
        scrollto(urlname+1);
        return;
    }

    selurl(urlname);
    selection->map=map;

    message("getting %s", urlstr(selection));
    esetcursor(&patientcurs);
    for(;;){
        if((fd=urlget(selection, post)) < 0){
            message("%r");
            break;
        }
        message("getting %s", selection->fullname);
        if(mothmode && !plumb)
            typ = -1;
        else if((typ = mimetotype(selection->contenttype)) < 0)
            typ = snooptype(fd);

        switch(typ){
        default:
            if(plumb){
                message("unknown file type");
                close(fd);
                break;
            }
            snprint(cmd, sizeof(cmd), "%s", urltofile(selection));
            if(eenter("Save to", cmd, sizeof(cmd), &mouse) <= 0){
                close(fd);
                break;
            }
            save(fd, cmd);
            break;
        case HTML:
            fd = pipeline(fd, "exec uhtml");
        case PLAIN:
            n=0;
            for(i=wwwtop-1; i>=0 && i!=(wwwtop-NWWW-1); i--){
                w = www(i);
                n += countpix(w->pix);
                if(n >= NPIXMB*1024*1024)
                    killpix(w);
            }
```

```
            w = www(i = wwwtop++);
            if(i >= NWWW){
                /* wait for the reader to finish the document */
                while(!w->finished && !w->alldone){
                    drawlock(0);
                    sleep(10);
                    drawlock(1);
                }
                freetext(w->text);
                freeform(w->form);
                freepix(w->pix);
                freeurl(w->url);
                memset(w, 0, sizeof(*w));
            }
            if(selection->map)
                w->url=copyurl(current->url);
            else
                w->url=copyurl(selection);
            w->finished = 0;
            w->alldone = 0;
            gettext(w, fd, typ);
            if(rfork(RFPROC|RFMEM|RFNOWAIT) == 0){
                for(;;){
                    sleep(1000);
                    if(w->finished || w->alldone)
                        break;
                    if(w->changed)
                        write(kickpipe[1], "C", 1);
                }
                _exits(0);
            }
            plinitlist(list, PACKN|FILLX, genwww, 8, doprev);
            if(defdisplay) pldraw(list, view);
            setcurrent(i, selection->tag);
            break;
        case GIF:
        case JPEG:
        case PNG:
        case BMP:
        case PAGE:
            filter(fd, "exec page -w");
            break;
        }
        break;
    }
    donecurs();
}
```

⟨*function* updtext*(mothra)* 177⟩≡ (181)

```
void updtext(Www *w){
    Rtext *t;
    Action *a;
    if(defdisplay && w->gottitle==0 && w->title[0]!='\0')
        pldraw(list, view);
    for(t=w->text;t;t=t->next){
        a=t->user;
        if(a){
            if(a->field)
                mkfieldpanel(t);
            a->field=0;
```

```
        }
    }
    if(w != current)
        return;
    w->yoffs=plgetpostextview(text);
    plinittextview(text, PACKE|EXPAND, Pt(0, 0), w->text, dolink);
    plsetpostextview(text, w->yoffs);
    pldraw(text, view);
}
```

⟨*function* finish*(mothra)* 178a⟩≡ (181)

```
void finish(Www *w){
    w->finished = 1;
    write(kickpipe[1], "F", 1);
}
```

⟨*function* mothon*(mothra)* 178b⟩≡ (181)

```
void
mothon(Www *w, int on)
{
    Rtext *t, *x;
    Action *a, *ap;

    if(w==0 || mothmode==on)
        return;
    if(mothmode = on)
        message("moth mode!");
    else
        message(mothra);
    /*
     * insert or remove artificial links to the href for
     * images that are also links
     */
    for(t=w->text;t;t=t->next){
        a=t->user;
        if(a == nil || a->image == nil)
            continue;
        if(a->link == nil){
            if(on)
                t->flags |= PL_HOT;
            else
                t->flags &= ~PL_HOT;
            continue;
        }
        x = t->next;
        if(on){
            t->next = nil;
            ap=emalloc(sizeof(Action));
            ap->link = strdup(a->link);
            plrtstr(&t->next, 0, 0, /*0,*/ t->font, strdup("->"), PL_HOT, ap);
            t->next->next = x;
        } else {
            if(x) {
                t->next = x->next;
                x->next = nil;
                freetext(x);
            }
        }
    }
    updtext(w);
```

```
    donecurs();
}
```

⟨*function* killpix*(mothra)* 179a⟩≡ (181)

```
void killpix(Www *w){
    Rtext *t;

    if(w==0 || !w->finished && !w->alldone)
        return;
    for(t=w->text; t; t=t->next)
        if(t->b && t->user)
            t->b=0;
    freepix(w->pix);
    w->pix=0;
    updtext(w);
}
```

⟨*function* snarf*(mothra)* 179b⟩≡ (181)

```
void snarf(Panel *p){
    if(p==0 || p==cmd){
        if(selection){
            plputsnarf(urlstr(selection));
            plsnarf(text);
        }else
            message("no url selected");
    }else
        plsnarf(p);
}
```

⟨*function* paste*(mothra)* 179c⟩≡ (181)

```
void paste(Panel *p){
    if(p==0) p=cmd;
    plpaste(p);
}
```

⟨*function* hit3*(mothra)* 179d⟩≡ (181)

```
void hit3(int button, int item){
    char buf[1024];
    char name[NNAME];
    char *s;
    Panel *swap;
    int fd;
    USED(button);
    switch(item){
    case 0:
        swap=root;
        root=alt;
        alt=swap;
        if(current)
            current->yoffs=plgetpostextview(text);
        swap=text;
        text=alttext;
        alttext=swap;
        defdisplay=!defdisplay;
        plpack(root, view->r);
        if(current){
            plinittextview(text, PACKE|EXPAND, Pt(0, 0), current->text, dolink);
            plsetpostextview(text, current->yoffs);
        }
        pldraw(root, view);
```

```
    break;
case 1:
    mothon(current, !mothmode);
    break;
case 2:
    snarf(plkbfocus);
    break;
case 3:
    paste(plkbfocus);
    break;
case 4:
    if(plkbfocus==nil || plkbfocus==cmd){
        if(text==nil || text->snarf==nil || selection==nil)
            return;
        if((s=text->snarf(text))==nil)
            s=smprint("%s", urlstr(selection));
    }else
        if((s=plkbfocus->snarf(plkbfocus))==nil)
            return;
    if((fd=plumbopen("send", OWRITE))<0){
        message("can't plumb");
        free(s);
        return;
    }
    plumbsendtext(fd, "mothra", nil, getwd(buf, sizeof buf), s);
    close(fd);
    free(s);
    break;
case 5:
    search();
    break;
case 6:
    if(!selection){
        message("no url selected");
        break;
    }
    snprint(name, sizeof(name), "%s/hit.html", mkhome());
    fd=open(name, OWRITE);
    if(fd==-1){
        fd=create(name, OWRITE, 0666);
        if(fd==-1){
            message("can't open %s", name);
            return;
        }
        fprint(fd, "<html><head><title>Hit List</title></head>\n");
        fprint(fd, "<body><h1>Hit list</h1>\n");
    }
    seek(fd, 0, 2);
    fprint(fd, "<p><a href=\"%s\">%s</a>\n", urlstr(selection), urlstr(selection));
    close(fd);
    break;
case 7:
    snprint(name, sizeof(name), "file:%s/hit.html", mkhome());
    geturl(name, -1, 1, 0);
    break;
case 8:
    if(confirm(3))
        exits(0);
    break;
}
```

```
}
```

⟨mothra/mothra.c 181⟩≡

```
/*
 * Trivial web browser
 */
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <event.h>
#include <keyboard.h>
#include <plumb.h>
#include <cursor.h>
#include <panel.h>
#include <regexp.h>

#include "mothra.h"
#include "rtext.h"

int debug=0;
int verbose=0;          /* -v flag causes html errors to be written to file-descriptor 2 */

int killimgs=0; /* should mothra kill images? */
int defdisplay=1;       /* is the default (initial) display visible? */
int visxbar=0;  /* horizontal scrollbar visible? */
int topxbar=0;  /* horizontal scrollbar at top? */
```

⟨*global* root*(mothra)* 159b⟩

⟨*global* alt*(mothra)* 159c⟩
⟨*global* alttext*(mothra)* 159d⟩
⟨*global* cmd*(mothra)* 159e⟩
⟨*global* cururl*(mothra)* 159f⟩
⟨*global* list*(mothra)* 160a⟩
⟨*global* msg*(mothra)* 160b⟩
⟨*global* menu3*(mothra)* 160c⟩

⟨*global* mothra*(mothra)* 160d⟩

```
Cursor patientcurs={
    0, 0,
    0x01, 0x80, 0x03, 0xC0, 0x07, 0xE0, 0x07, 0xe0,
    0x07, 0xe0, 0x07, 0xe0, 0x03, 0xc0, 0x0F, 0xF0,
    0x1F, 0xF8, 0x1F, 0xF8, 0x1F, 0xF8, 0x1F, 0xF8,
    0x0F, 0xF0, 0x1F, 0xF8, 0x3F, 0xFC, 0x3F, 0xFC,

    0x01, 0x80, 0x03, 0xC0, 0x07, 0xE0, 0x04, 0x20,
    0x04, 0x20, 0x06, 0x60, 0x02, 0x40, 0x0C, 0x30,
    0x10, 0x08, 0x14, 0x08, 0x14, 0x28, 0x12, 0x28,
    0x0A, 0x50, 0x16, 0x68, 0x20, 0x04, 0x3F, 0xFC,
};
Cursor confirmcursor={
    0, 0,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

    0x00, 0x0E, 0x07, 0x1F, 0x03, 0x17, 0x73, 0x6F,
```

```
    0xFB, 0xCE, 0xDB, 0x8C, 0xDB, 0xC0, 0xFB, 0x6C,
    0x77, 0xFC, 0x00, 0x00, 0x00, 0x01, 0x00, 0x03,
    0x94, 0xA6, 0x63, 0x3C, 0x63, 0x18, 0x94, 0x90,
};
Cursor readingcurs={
    -10, -3,
    0x00, 0x00, 0x00, 0x00, 0x0F, 0xF0, 0x0F, 0xF0,
    0x0F, 0xF0, 0x0F, 0xF0, 0x0F, 0xF0, 0x1F, 0xF0,
    0x3F, 0xF0, 0x7F, 0xF0, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFB, 0xFF, 0xF3, 0xFF, 0x00, 0x00, 0x00, 0x00,

    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xE0,
    0x07, 0xE0, 0x01, 0xE0, 0x03, 0xE0, 0x07, 0x60,
    0x0E, 0x60, 0x1C, 0x00, 0x38, 0x00, 0x71, 0xB6,
    0x61, 0xB6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
Cursor mothcurs={
    {-7, -7},
    {0x00, 0x00, 0x60, 0x06, 0xf8, 0x1f, 0xfc, 0x3f,
     0xfe, 0x7f, 0xff, 0xff, 0x7f, 0xfe, 0x7f, 0xfe,
     0x7f, 0xfe, 0x3f, 0xfc, 0x3f, 0xfc, 0x1f, 0xf8,
     0x1f, 0xf8, 0x0e, 0x70, 0x0c, 0x30, 0x00, 0x00, },
    {0x00, 0x00, 0x00, 0x00, 0x60, 0x06, 0x58, 0x1a,
     0x5c, 0x3a, 0x64, 0x26, 0x27, 0xe4, 0x37, 0xec,
     0x37, 0xec, 0x17, 0xe8, 0x1b, 0xd8, 0x0e, 0x70,
     0x0c, 0x30, 0x04, 0x20, 0x00, 0x00, 0x00, 0x00, }
};
```

⟨*global* `current` *(mothra)* 160e⟩
⟨*global* `selection` *(mothra)* 160f⟩

```
int mothmode;
int kickpipe[2];

void docmd(Panel *, char *);
void doprev(Panel *, int, int);
char *urlstr(Url *);
void setcurrent(int, char *);
char *genwww(Panel *, int);
void updtext(Www *);
void dolink(Panel *, int, Rtext *);
void hit3(int, int);
void mothon(Www *, int);
void killpix(Www *w);

// renamed to avoid conflict with event.h typedef buttons
char *buttons_[]={
    "alt display",
    "moth mode",
    "snarf",
    "paste",
    "plumb",
    "search",
    "save hit",
    "hit list",
    "exit",
    0
};
```

```
int wwwtop=0;
```

⟨*function* `www`*(mothra)* 160g⟩
⟨*function* `nwww`*(mothra)* 160h⟩

⟨*function* `subpanel`*(mothra)* 160i⟩
⟨*function* `adjkb`*(mothra)* 160j⟩

⟨*function* `scrollpanel`*(mothra)* 161a⟩

⟨*function* `sidescroll`*(mothra)* 161b⟩

⟨*function* `mkpanels`*(mothra)* 161c⟩
```
int cohort = -1;
```
⟨*function* `killcohort`*(mothra)* 162a⟩
⟨*function* `catch`*(mothra)* 162b⟩
⟨*function* `dienow`*(mothra)* 162c⟩

⟨*function* `mkhome`*(mothra)* 162d⟩

⟨*function* `donecurs`*(mothra)* 163a⟩

⟨*function* `drawlock`*(mothra)* 163b⟩

```
void scrollto(char *tag);
void search(void);

extern char *mtpt; /* url */
```

⟨*function* `main`*(mothra)* 163c⟩

⟨*function* `confirm`*(mothra)* 166a⟩

⟨*function* `message`*(mothra)* 166b⟩

⟨*function* `htmlerror`*(mothra)* 166c⟩

⟨*function* `eresized`*(mothra)* 167a⟩

⟨*function* `emalloc`*(mothra)* 167b⟩

⟨*function* `nstrcpy`*(mothra)* 167c⟩

⟨*function* `genwww`*(mothra)* 167d⟩

⟨*function* `scrollto`*(mothra)* 168a⟩

⟨*function* `setcurrent`*(mothra)* 168b⟩
⟨*function* `arg`*(mothra)* 168c⟩
⟨*function* `save`*(mothra)* 168d⟩
⟨*function* `screendump`*(mothra)* 169a⟩

⟨*function* `urltofile`*(mothra)* 169b⟩

⟨*function* `docmd`*(mothra)* 170⟩

⟨*function* `regerror`*(mothra)* 171a⟩

⟨*function* `search`*(mothra)* 171b⟩

⟨*function* `hiturl`*(mothra)* 172a⟩

⟨*function* `doprev`*(mothra)* 172b⟩

⟨*function* `dolink`*(mothra)* 172c⟩

⟨*function* `filter`*(mothra)* 173a⟩
⟨*function* `gettext`*(mothra)* 173b⟩

⟨*function* `freetext`*(mothra)* 173c⟩

⟨*function* `dupfds`*(mothra)* 174a⟩

⟨*function* `pipeline`*(mothra)* 174b⟩

⟨*function* `urlstr`*(mothra)* 175a⟩

⟨*function* `copyurl`*(mothra)* 175b⟩

⟨*function* `freeurl`*(mothra)* 175c⟩

⟨*function* `seturl`*(mothra)* 175d⟩

⟨*function* `selurl`*(mothra)* 175e⟩

⟨*function* `geturl`*(mothra)* 176⟩
⟨*function* `updtext`*(mothra)* 177⟩

⟨*function* `finish`*(mothra)* 178a⟩

⟨*function* `mothon`*(mothra)* 178b⟩

⟨*function* `killpix`*(mothra)* 179a⟩
⟨*function* `snarf`*(mothra)* 179b⟩
⟨*function* `paste`*(mothra)* 179c⟩
⟨*function* `hit3`*(mothra)* 179d⟩

mothra/rdhtml.c

⟨*global* `fontlist`*(mothra)* 184⟩≡ (206)

```
struct Fontdata{
    char *name;
    Font *font;
    int space;
}fontlist[4][4]={
    "dejavusans/unicode.12", 0, 0,
    "dejavusans/unicode.12", 0, 0,
    "dejavusans/unicode.14", 0, 0,
    "dejavusans/unicode.16", 0, 0,

    "dejavusansit/unicode.12", 0, 0,
    "dejavusansit/unicode.12", 0, 0,
    "dejavusansit/unicode.14", 0, 0,
    "dejavusansit/unicode.16", 0, 0,

    "dejavusansbd/unicode.12", 0, 0,
    "dejavusansbd/unicode.12", 0, 0,
    "dejavusansbd/unicode.14", 0, 0,
    "dejavusansbd/unicode.16", 0, 0,
```

```
    "terminus/unicode.12", 0, 0,
    "terminus/unicode.14", 0, 0,
    "terminus/unicode.16", 0, 0,
    "terminus/unicode.18", 0, 0,

/* original */
/*
    "lucidasans/unicode.7", 0, 0,
    "lucidasans/unicode.8", 0, 0,
    "lucidasans/unicode.10", 0, 0,
    "lucidasans/unicode.13", 0, 0,

    "lucidasans/italicunicode.7", 0, 0,
    "lucidasans/italicunicode.8", 0, 0,
    "lucidasans/italicunicode.10", 0, 0,
    "lucidasans/italicunicode.13", 0, 0,

    "lucidasans/boldunicode.7", 0, 0,
    "lucidasans/boldunicode.8", 0, 0,
    "lucidasans/boldunicode.10", 0, 0,
    "lucidasans/boldunicode.13", 0, 0,

    "lucidasans/typeunicode.7", 0, 0,
    "pelm/unicode.8", 0, 0,
    "lucidasans/typeunicode.12", 0, 0,
    "lucidasans/typeunicode.16", 0, 0,
*/
};
```

⟨*global* `links` *(mothra)* 185a⟩≡ (206)

```
static struct{
    char *prefix;
    int len;
}links[]={
    {"http://", 7},
    {"https://", 8},
    {"gemini://", 9},
    {"ftp://", 6},
};
```

⟨*function* `pl_whichfont` *(mothra)* 185b⟩≡ (206)

```
Font *pl_whichfont(int f, int s, int *space){
    char name[NNAME];

    assert(f >= 0 && f < 4);
    assert(s >= 0 && s < 4);

    if(fontlist[f][s].font==0){
        snprint(name, sizeof(name), "/lib/font/bit/%s.font", fontlist[f][s].name);
        fontlist[f][s].font=openfont(display, name);
        if(fontlist[f][s].font==0) fontlist[f][s].font=font;
        fontlist[f][s].space=stringwidth(fontlist[f][s].font, "0");
    }
    if(space)
        *space = fontlist[f][s].space;
    return fontlist[f][s].font;
}
```

⟨*function* `getfonts` *(mothra)* 186a⟩≡ (206)

```
void getfonts(void){
    int f, s;
    for(f=0;f!=4;f++)
        for(s=0;s!=4;s++)
            pl_whichfont(f, s, nil);
}
```

⟨*function* `pl_pushstate` *(mothra)* 186b⟩≡ (206)

```
void pl_pushstate(Hglob *g, int t){
    ++g->state;
    if(g->state==&g->stack[NSTACK]){
        htmlerror(g->name, g->lineno, "stack overflow at <%s>", tag[t].name);
        --g->state;
    }
    g->state[0]=g->state[-1];
    g->state->tag=t;

    if(g->state->name)
        g->state->name = strdup(g->state->name);
    if(g->state->link)
        g->state->link = strdup(g->state->link);
    if(g->state->image)
        g->state->image = strdup(g->state->image);
}
```

⟨*function* `pl_popstate` *(mothra)* 186c⟩≡ (206)

```
void pl_popstate(Stack *state){
    free(state->name);
    state->name=0;
    free(state->link);
    state->link=0;
    free(state->image);
    state->image=0;
}
```

⟨*function* `pl_linespace` *(mothra)* 186d⟩≡ (206)

```
void pl_linespace(Hglob *g){
    plrtbitmap(&g->dst->text, 1000000, 0, /*0,*/ linespace, 0, 0);
    g->para=0;
    g->linebrk=0;
}
```

⟨*function* `strtolength` *(mothra)* 186e⟩≡ (206)

```
int strtolength(Hglob *g, int dir, char *str){
    double f;
    Point p;

    f = atof(str);
    if(cistrstr(str, "%"))
        return 0;
    if(cistrstr(str, "em")){
        p=stringsize(pl_whichfont(g->state->font, g->state->size, nil), "M");
        return floor(f*((dir==HORIZ) ? p.x : p.y));
    }
    return floor(f);
}
```

⟨*function* pl_htmloutput*(mothra)* 187⟩≡ (206)

```
void pl_htmloutput(Hglob *g, int nsp, char *s, Field *field){
    Font *f;
    int space, indent, flags, voff;
    Action *ap;
    if(g->state->tag==Tag_title
/*      || g->state->tag==Tag_textarea */
    || g->state->tag==Tag_select){
        if(s){
            if(g->tp!=g->text && g->tp!=g->etext && g->tp[-1]!=' ')
                *g->tp++=' ';
            while(g->tp!=g->etext && *s) *g->tp++=*s++;
            if(g->state->tag==Tag_title) g->dst->changed=1;
            *g->tp='\0';
        }
        return;
    }
    voff = 0;
    f=pl_whichfont(g->state->font, g->state->size, &space);
    if(g->state->sub){
        voff = g->state->sub * f->ascent / 2;
        g->state->size = SMALL;
        f=pl_whichfont(g->state->font, g->state->size, &space);
    }
    indent=g->state->margin;
    if(g->para){
        space=1000000;
        indent+=g->state->indent;
    }
    else if(g->linebrk)
        space=1000000;
    else if(nsp<=0)
        space=0;
    if(g->state->image==0 && g->state->link==0 && g->state->name==0 && field==0)
        ap=0;
    else{
        ap=emalloc(sizeof(Action));
        if(g->state->image)
            ap->image = strdup(g->state->image);
        if(g->state->link)
            ap->link = strdup(g->state->link);
        if(g->state->name)
            ap->name = strdup(g->state->name);
        ap->ismap=g->state->ismap;
        ap->width=g->state->width;
        ap->height=g->state->height;
        ap->field=field;
    }
    if(space<0) space=0;
    if(indent<0) indent=0;
    if(g->state->pre && s[0]=='\t'){
        space=0;
        while(s[0]=='\t'){
            space++;
            s++;
        }
        space=PL_TAB|space;
        if(g->linebrk){
            indent=space;
            space=1000000;
```

```
        }
    }
    flags = 0;
    if(g->state->link)
        flags |= PL_HOT;
    //if(g->state->strike)
    //  flags |= PL_STR;
    plrtstr(&g->dst->text, space, indent, /*voff,*/ f, strdup(s), flags, ap);
    g->para=0;
    g->linebrk=0;
    g->dst->changed=1;
}
```

⟨*function* pl_bread*(mothra)* 188a⟩≡ (206)

```
/*
 * Buffered read, no translation
 * Save in cache.
 */
int pl_bread(Hglob *g){
    int n, c;
    char err[1024];
    if(g->hbufp==g->ehbuf){
        n=read(g->hfd, g->hbuf, NHBUF);
        if(n<=0){
            if(n<0){
                snprint(err, sizeof(err), "%r reading %s", g->name);
                pl_htmloutput(g, 1, err, 0);
            }
            g->heof=1;
            return EOF;
        }
        g->hbufp=g->hbuf;
        g->ehbuf=g->hbuf+n;
    }
    c=*g->hbufp++&255;
    if(c=='\n') g->lineno++;
    return c;
}
```

⟨*function* pl_readc*(mothra)* 188b⟩≡ (206)

```
/*
 * Read a character, translating \r\n, \n\r, \r and \n into \n
 * convert to runes.
 */
int pl_readc(Hglob *g){
    static int peek=-1;
    char crune[UTFmax+1];
    int c, n;
    Rune r;

    if(peek!=-1){
        c=peek;
        peek=-1;
    }
    else
        c=pl_bread(g);
    if(c=='\r'){
        c=pl_bread(g);
        if(c!='\n') peek=c;
        return '\n';
```

```
    }
    if(c=='\n'){
        c=pl_bread(g);
        if(c!='\r') peek=c;
        return '\n';
    }

    if(c < Runeself)
        return c;

    crune[0]=c;
    for (n=1; n<=sizeof(crune); n++){
        if(fullrune(crune, n)){
            chartorune(&r, crune);
            return r;
        }
        c=pl_bread(g);
        if(c==EOF)
            return EOF;
        crune[n]=c;
    }
    return c;
}
```

⟨*function* pl_putback*(mothra)* 189a⟩≡ (206)

```
void pl_putback(Hglob *g, int c){
    if(g->npeekc==NPEEKC) htmlerror(g->name, g->lineno, "too much putback!");
    else if(c!=EOF) g->peekc[g->npeekc++]=c;
}
```

⟨*function* pl_nextc*(mothra)* 189b⟩≡ (206)

```
int pl_nextc(Hglob *g){
    int c;

    if(g->heof) return EOF;
    if(g->npeekc!=0) return g->peekc[--g->npeekc];
    c=pl_readc(g);
    if(c=='<'){
        c=pl_readc(g);
        if(c=='/'){
            c=pl_readc(g);
            pl_putback(g, c);
            pl_putback(g, '/');
            if('a'<=c && c<='z' || 'A'<=c && c<='Z') return STAG;
            return '<';
        }
        pl_putback(g, c);
        if(c=='!' || 'a'<=c && c<='z' || 'A'<=c && c<='Z' || c=='?') return STAG;
        return '<';
    }
    if(c=='>') return ETAG;
    return c;
}
```

⟨*function* unquot*(mothra)* 189c⟩≡ (206)

```
char *unquot(char *src){
    char *e, *dst;
    int len;

    e=0;
```

```
    while(*src && strchr(" \t\r\n", *src))
        src++;
    if(*src=='\'' || *src=='"'){
        e=strrchr(src+1, *src);
        src++;
    }
    if(e==0) e=strchr(src, 0);
    len=e-src;
    dst = emalloc(len+1);
    memmove(dst, src, len);
    dst[len]=0;
    return dst;
}
```

⟨*function* `alnumchar` *(mothra)* 190a⟩≡ (206)

```
int alnumchar(int c){
    return 'a'<=c && c<='z' || 'A'<=c && c<='Z' || '0'<=c && c<='9';
}
```

⟨*function* `entchar` *(mothra)* 190b⟩≡ (206)

```
int entchar(int c){
    return c=='#' || alnumchar(c);
}
```

⟨*function* `linkify` *(mothra)* 190c⟩≡ (206)

```
/* return url if text token looks like a hyperlink */
char *linkify(char *s){
    int i;
    if(s == 0 && s[0] == 0)
        return 0;
    for(i = 0; i < nelem(links); i++)
        if(!cistrncmp(s, links[i].prefix, links[i].len))
            return strdup(s);
    if(!cistrncmp(s, "www.", 4)){
        int d, i;

        d = 1;
        for(i=4; s[i]; i++){
            if(s[i] == '.'){
                if(s[i-1] == '.')
                    return 0;
                d++;
            } else if(!alnumchar(s[i]))
                break;
        }
        if(d >= 2)
            return smprint("http://%s", s);
    }
    return 0;
}
```

⟨*function* `pl_rmentities` *(mothra)* 190d⟩≡ (206)

```
/*
 * remove entity references, in place.
 * Potential bug:
 *      This doesn't work if removing an entity reference can lengthen the string!
 *      Fortunately, this doesn't happen.
 */
void pl_rmentities(Hglob *, char *s){
    char *t, *u, c, svc;
```

```
    t=s;
    do{
        c=*s++;
        if(c=='&'
        && ((*s=='#' && strchr("0123456789Xx", s[1]))
          || 'a'<=*s && *s<='z'
          || 'A'<=*s && *s<='Z')){
            u=s;
            while(entchar(*s)) s++;
            svc=*s;
            *s = 0;
            if(svc==';') s++;
            if(strcmp(u, "lt") == 0)
                *t++='<';
            else if(strcmp(u, "gt") == 0)
                *t++='>';
            else if(strcmp(u, "quot") == 0)
                *t++='"';
            else if(strcmp(u, "apos") == 0)
                *t++='\'';
            else if(strcmp(u, "amp") == 0)
                *t++='&';
            else {
                if(svc==';') s--;
                *s=svc;
                *t++='&';
                while(u<s)
                    *t++=*u++;
            }
        }
        else if((uchar)c == 0xc2 && (uchar)*s == 0xad)
            s++; /* ignore soft hyphens */
        else
            *t++=c;
    }while(c);
}
```

⟨*function* `pl_whitespace`*(mothra)* 191a⟩≡ (206)

```
/*
 * Skip over white space
 */
char *pl_whitespace(char *s){
    while(*s==' ' || *s=='\t' || *s=='\n' || *s=='\r') s++;
    return s;
}
```

⟨*function* `pl_word`*(mothra)* 191b⟩≡ (206)

```
/*
 * Skip over HTML word
 */
char *pl_word(char *s){
    if ('a'<=*s && *s<='z' || 'A'<=*s && *s<='Z') {
        s++;
        while('a'<=*s && *s<='z' || 'A'<=*s && *s<='Z' || '0'<=*s && *s<='9' ||
            *s=='-' || *s=='.' || *s==':') s++;
    }
    return s;
}
```

⟨*function* `pl_quote` *(mothra)* 192a⟩≡ (206)

```
/*
 * Skip to matching quote
 */
char *pl_quote(char *s){
    char q;
    q=*s++;
    while(*s!=q && *s!='\0') s++;
    return s;
}
```

⟨*function* `pl_dnl` *(mothra)* 192b⟩≡ (206)

```
void pl_dnl(char *s){
    char *t;
    for(t=s;*s;s++) if(*s!='\r' && *s!='\n') *t++=*s;
    *t='\0';
}
```

⟨*function* `pl_tagparse` *(mothra)* 192c⟩≡ (206)

```
void pl_tagparse(Hglob *g, char *str){
    char *s, *t, *name, c;
    Pair *ap;
    Tag *tagp;
    g->tag=Tag_end;
    ap=g->attr;
    if(str[0]=='!'){    /* test should be strncmp(str, "!--", 3)==0 */
        g->tag=Tag_comment;
        ap->name=0;
        return;
    }
    if(str[0]=='/') str++;
    name=str;
    s=pl_word(str);
    if(*s!='/' && *s!=' ' && *s!='\n' && *s!='\t' && *s!='\0'){
        htmlerror(g->name, g->lineno, "bad tag name in %s", str);
        ap->name=0;
        return;
    }
    if(*s!='\0') *s++='\0';
    for(t=name;t!=s;t++) if('A'<=*t && *t<='Z') *t+='a'-'A';
    /*
     * Binary search would be faster here
     */
    for(tagp=tag;tagp->name;tagp++) if(strcmp(name, tagp->name)==0) break;
    g->tag=tagp-tag;
    if(g->tag==Tag_end) htmlerror(g->name, g->lineno, "no tag %s", name);
    for(;;){
        s=pl_whitespace(s);
        if(*s=='\0'){
            ap->name=0;
            return;
        }
        ap->name=s;
        s=pl_word(s);
        t=pl_whitespace(s);
        c=*t;
        *s='\0';
        for(s=ap->name;*s;s++) if('A'<=*s && *s<='Z') *s+='a'-'A';
        if(c=='='){
            s=pl_whitespace(t+1);
```

```
            if(*s=='\'' || *s=='"'){
                ap->value=s+1;
                s=pl_quote(s);
                if(*s=='\0'){
                    htmlerror(g->name, g->lineno,
                        "No terminating quote in rhs of attribute %s",
                        ap->name);
                    ap->name=0;
                    return;
                }
                *s++='\0';
                pl_dnl(ap->value);
            }
            else{
                /* read up to white space or > */
                ap->value=s;
                while(*s!=' ' && *s!='\t' && *s!='\n' && *s!='\0') s++;
                if(*s!='\0') *s++='\0';
            }
            pl_rmentities(g, ap->value);
        }
        else{
            if(c!='\0') s++;
            ap->value="";
        }
        if(ap==&g->attr[NATTR-1])
            htmlerror(g->name, g->lineno, "too many attributes!");
        else ap++;
    }
}
```

⟨*function* `pl_getcomment` *(mothra)* 193⟩≡ (206)

```
int pl_getcomment(Hglob *g){
    int c;
    if((c=pl_nextc(g))=='-' && (c=pl_nextc(g))=='-'){
        /* <!-- eats everything until --> or EOF */
        for(;;){
            while((c=pl_nextc(g))!='-' && c!=EOF)
                ;
            if(c==EOF)
                break;
            if(pl_nextc(g)=='-'){
                while((c=pl_nextc(g))=='-')
                    ;
                if(c==ETAG || c==EOF)
                    break;
            }
        }
    } else {
        /* <! eats everything until > or EOF */
        while(c!=ETAG && c!=EOF)
            c=pl_nextc(g);
    }
    if(c==EOF)
        htmlerror(g->name, g->lineno, "EOF in comment");
    g->tag=Tag_comment;
    g->attr->name=0;
    g->token[0]='\0';
    return TAG;
}
```

⟨*function* `lrunetochar` *(mothra)* 194a⟩≡ (206)

```
int lrunetochar(char *p, int v)
{
    Rune r;

    r=v;
    return runetochar(p, &r);
}
```

⟨*function* `pl_getscript` *(mothra)* 194b⟩≡ (206)

```
int pl_getscript(Hglob *g){
    char *tokp, *t;
    int c;
    tokp = g->token;
    *tokp++ = '<';
    while((c=pl_nextc(g)) != EOF){
        if(c==STAG || c==' ' || c=='\t' || c=='\n'){
            pl_putback(g, c);
            break;
        }
        if(c==ETAG) c='>';
        tokp += lrunetochar(tokp, c);
        if(c==0 || c=='>' || tokp >= &g->token[NTOKEN-UTFmax-1])
            break;
    }
    *tokp = '\0';
    t = tag[g->state->tag].name;
    if(g->token[1] == '/' && cistrncmp(g->token+2, t, strlen(t)) == 0){
        g->tag=g->state->tag;
        g->attr->name=0;
        return ENDTAG;
    }
    pl_rmentities(g, g->token);
    g->nsp=g->spacc;
    g->spacc=0;
    return TEXT;
}
```

⟨*function* `pl_gettag` *(mothra)* 194c⟩≡ (206)

```
/*
 * Read a start or end tag -- the caller has read the initial <
 */
int pl_gettag(Hglob *g){
    char *tokp;
    int c, q;
    if(g->state->isscript)
        return pl_getscript(g);
    if((c=pl_nextc(g))=='!' || c=='?')
        return pl_getcomment(g);
    pl_putback(g, c);
    q = 0;
    tokp=g->token;
    while((c=pl_nextc(g))!=EOF){
        if(c == '=' && q == 0)
            q = '=';
        else if(c == '\'' || c == '"'){
            if(q == '=')
                q = c;
            else if(q == c)
                q = 0;
```

```
        }
        else if(c == ETAG && q != '\'' && q != '"')
            break;
        else if(q == '=' && c != ' ' && c != '\t' && c != '\n')
            q = 0;
        if(tokp < &g->token[NTOKEN-UTFmax-1])
            tokp += lrunetochar(tokp, c);
    }
    *tokp='\0';
    if(c==EOF) htmlerror(g->name, g->lineno, "EOF in tag");
    pl_tagparse(g, g->token);
    if(g->token[0]!='/') return TAG;
    if(g->attr[0].name!=0)
        htmlerror(g->name, g->lineno, "end tag should not have attributes");
    return ENDTAG;
}
```

⟨*function* `pl_gettoken`*(mothra)* 195⟩≡ (206)

```
/*
 * The next token is a tag, an end tag or a sequence of non-white
 * characters. If inside <pre>, single newlines are converted to <br>,
 * double newlines are converted to <p> and spaces are preserved.
 * Otherwise, spaces and newlines are noted and discarded.
 */
int pl_gettoken(Hglob *g){
    char *tokp;
    int c;
    if(g->state->pre) switch(c=pl_nextc(g)){
    case STAG: return pl_gettag(g);
    case EOF: return EOF;
    case '\n':
        switch(c=pl_nextc(g)){
        case '\n':
            pl_tagparse(g, "p");
            return TAG;
        default:
            pl_tagparse(g, "br");
            pl_putback(g, c);
            return TAG;
        }
    default:
        tokp=g->token;
        while(c=='\t'){
            if(tokp < &g->token[NTOKEN-UTFmax-1]) tokp += lrunetochar(tokp, c);
            c=pl_nextc(g);
        }
        while(c!='\t' && c!='\n' && c!=STAG && c!=EOF){
            if(c==ETAG) c='>';
            if(tokp < &g->token[NTOKEN-UTFmax-1]) tokp += lrunetochar(tokp, c);
            c=pl_nextc(g);
        }
        *tokp='\0';
        pl_rmentities(g, g->token);
        pl_putback(g, c);
        g->nsp=0;
        g->spacc=0;
        return TEXT;
    }
    while((c=pl_nextc(g))==' ' || c=='\t' || c=='\n')
        if(g->spacc!=-1)
```

```
            g->spacc++;
        switch(c){
        case STAG: return pl_gettag(g);
        case EOF: return EOF;
        default:
            tokp=g->token;
            do{
                if(c==ETAG) c='>';
                if(tokp < &g->token[NTOKEN-UTFmax-1]) tokp += lrunetochar(tokp, c);
                c=pl_nextc(g);
            }while(c!=' ' && c!='\t' && c!='\n' && c!=STAG && c!=EOF);
            *tokp='\0';
            pl_rmentities(g, g->token);
            pl_putback(g, c);
            g->nsp=g->spacc;
            g->spacc=0;
            return TEXT;
        }
    }
```

⟨*function* pl_getattr*(mothra)* 196a⟩≡ (206)

```
    char *pl_getattr(Pair *attr, char *name){
        for(;attr->name;attr++)
            if(strcmp(attr->name, name)==0)
                return attr->value;
        return 0;
    }
```

⟨*function* pl_hasattr*(mothra)* 196b⟩≡ (206)

```
    int pl_hasattr(Pair *attr, char *name){
        for(;attr->name;attr++)
            if(strcmp(attr->name, name)==0)
                return 1;
        return 0;
    }
```

⟨*function* plaintext*(mothra)* 196c⟩≡ (206)

```
    void plaintext(Hglob *g){
        char line[NLINE];
        char *lp, *elp;
        int c;
        g->state->font=CWIDTH;
        g->state->size=NORMAL;
        g->state->sub = 0;
        elp=&line[NLINE-UTFmax-1];
        lp=line;
        for(;;){
            c=pl_readc(g);
            if(c==EOF) break;
            if(c=='\n' || lp>=elp){
                *lp='\0';
                g->linebrk=1;
                pl_htmloutput(g, 0, line, 0);
                lp=line;
            }
            if(c=='\t'){
                do *lp++=' '; while(lp<elp && utfnlen(line, lp-line)%8!=0);
            }
            else if(c!='\n')
                lp += lrunetochar(lp, c);
```

```
    }
    if(lp!=line){
        *lp='\0';
        g->linebrk=1;
        pl_htmloutput(g, 0, line, 0);
    }
}
```

⟨*function* plrdplain*(mothra)* 197a⟩≡ (206)

```
void plrdplain(char *name, int fd, Www *dst){
    Hglob g;
    g.state=g.stack;
    g.state->tag=Tag_html;
    g.state->font=CWIDTH;
    g.state->size=NORMAL;
    g.state->sub=0;
    g.state->pre=0;
    g.state->image=0;
    g.state->link=0;
    g.state->name=0;
    g.state->margin=0;
    g.state->indent=20;
    g.state->ismap=0;
    g.state->isscript=0;
    g.state->strike=0;
    g.state->width=0;
    g.state->height=0;
    g.dst=dst;
    g.hfd=fd;
    g.name=name;
    g.ehbuf=g.hbufp=g.hbuf;
    g.npeekc=0;
    g.heof=0;
    g.lineno=1;
    g.linebrk=1;
    g.para=0;
    g.text=dst->title;
    g.tp=g.text;
    g.etext=g.text+NTITLE-1;
    g.spacc=0;
    g.form=0;
    nstrcpy(g.text, name, NTITLE);
    plaintext(&g);
    finish(dst);
}
```

⟨*function* plrdhtml*(mothra)* 197b⟩≡ (206)

```
void plrdhtml(char *name, int fd, Www *dst, int killimgs){
    int tagerr;
    Stack *sp;
    char buf[20];
    char *str;
    Hglob g;

    g.state=g.stack;
    g.state->tag=Tag_html;
    g.state->font=ROMAN;
    g.state->size=NORMAL;
    g.state->sub=0;
    g.state->pre=0;
```

```
g.state->image=0;
g.state->link=0;
g.state->name=0;
g.state->margin=0;
g.state->indent=25;
g.state->ismap=0;
g.state->isscript=0;
g.state->strike=0;
g.state->width=0;
g.state->height=0;
g.dst=dst;
g.hfd=fd;
g.name=name;
g.ehbuf=g.hbufp=g.hbuf;
g.npeekc=0;
g.heof=0;
g.lineno=1;
g.linebrk=1;
g.para=0;
g.text=dst->title;
g.tp=g.text;
g.etext=g.text+NTITLE-1;
dst->title[0]='\0';
g.spacc=0;
g.form=0;

for(;;) switch(pl_gettoken(&g)){
case TAG:
    switch(tag[g.tag].action){
    case OPTEND:
        for(sp=g.state;sp!=g.stack && sp->tag!=g.tag;--sp);
        if(sp->tag!=g.tag)
            pl_pushstate(&g, g.tag);
        else
            for(;g.state!=sp;--g.state){
                if(tag[g.state->tag].action!=OPTEND)
                    htmlerror(g.name, g.lineno,
                        "end tag </%s> missing",
                        tag[g.state->tag].name);
                pl_popstate(g.state);
            }
        break;
    case END:
        pl_pushstate(&g, g.tag);
        break;
    }
    str=pl_getattr(g.attr, "id");
    if(str && *str){
        char *swap;

        swap = g.state->name;
        g.state->name = str;
        pl_htmloutput(&g, 0, "", 0);
        g.state->name = swap;
    }
    switch(g.tag){
    default:
        htmlerror(g.name, g.lineno,
            "unimplemented tag <%s>", tag[g.tag].name);
        break;
```

```
case Tag_end:   /* unrecognized start tag */
    break;
case Tag_img:
case Tag_image:
    str=pl_getattr(g.attr, "src");
    if(str && *str){
        free(g.state->image);
        g.state->image = strdup(str);
        //Url x;
        //seturl(&x, str, "http://iwp9.org");
        //// TODO: check return code
        //urlresolve(&x);
        ////free(str?)
        //g.state->image = strdup(urlstr(&x));
    } else {
        Pair *a;

        /*
         * hack to emulate javascript that rewrites some attribute
         * into src= after page got loaded. just look for some
         * attribute that looks like a url.
         */
        for(a = g.attr; a->name; a++){
            if(strcmp(a->name, "longdesc") == 0)
                continue;
            if(str = linkify(a->value)){
                free(g.state->image);
                g.state->image = str;
                break;
            }
        }
    }
    g.state->ismap=pl_hasattr(g.attr, "ismap");
    str=pl_getattr(g.attr, "width");
    if(str && *str)
        g.state->width=strtolength(&g, HORIZ, str);
    str=pl_getattr(g.attr, "height");
    if(str && *str)
        g.state->height=strtolength(&g, VERT, str);
    str=pl_getattr(g.attr, "alt");
    if(str==0 || *str == 0){
        if(g.state->image)
            str=g.state->image;
        else
            str="[[image]]";
    }
    pl_htmloutput(&g, 0, str, 0);
    free(g.state->image);
    g.state->image=0;
    g.state->ismap=0;
    g.state->width=0;
    g.state->height=0;
    break;
case Tag_plaintext:
    g.spacc=0;
    plaintext(&g);
    break;
case Tag_comment:
case Tag_html:
case Tag_link:
```

```
case Tag_nextid:
case Tag_table:
    break;
case Tag_tr:
    g.spacc=0;
    g.linebrk=1;
    break;
case Tag_th:
    g.state->font=BOLD;
case Tag_td:
    g.spacc++;
    break;
case Tag_base:
    str=pl_getattr(g.attr, "href");
    if(str && *str){
        seturl(g.dst->url, str, g.dst->url->fullname);
        nstrcpy(g.dst->url->fullname, str, sizeof(g.dst->url->fullname));
        /* base should be a full url, but it often isnt so have to resolve */
        urlresolve(g.dst->url);
    }
    break;
case Tag_a:
    str=pl_getattr(g.attr, "name");
    if(str && *str){
        free(g.state->name);
        g.state->name = strdup(str);
    }
    pl_htmloutput(&g, 0, "", 0);
    str=pl_getattr(g.attr, "href");
    if(str && *str){
        free(g.state->link);
        g.state->link = strdup(str);
    }
    break;
case Tag_meta:
    if((str=pl_getattr(g.attr, "http-equiv"))==0)
        break;
    if(cistrcmp(str, "refresh"))
        break;
    if((str=pl_getattr(g.attr, "content"))==0)
        break;
    if((str=strchr(str, '='))==0)
        break;
    str++;
    pl_htmloutput(&g, 0, "[refresh: ", 0);
    free(g.state->link);
    g.state->link=unquot(str);
    pl_htmloutput(&g, 0, g.state->link, 0);
    free(g.state->link);
    g.state->link=0;
    pl_htmloutput(&g, 0, "]", 0);
    g.linebrk=1;
    g.spacc=0;
    break;
case Tag_source:
case Tag_video:
case Tag_audio:
case Tag_embed:
case Tag_frame:
case Tag_iframe:
```

```
    snprint(buf, sizeof(buf), "[%s: ", tag[g.tag].name);
    pl_htmloutput(&g, 0, buf, 0);
    str=pl_getattr(g.attr, "src");
    if(str && *str){
        free(g.state->link);
        g.state->link = strdup(str);
    }
    str=pl_getattr(g.attr, "name");
    if(str && *str){
        free(g.state->name);
        g.state->name = strdup(str);
    } else if(g.state->link)
        str = g.state->link;
    else
        str = "";
    pl_htmloutput(&g, 0, str, 0);
    free(g.state->link);
    g.state->link=0;
    free(g.state->name);
    g.state->name=0;
    pl_htmloutput(&g, 0, "]", 0);
    g.linebrk=1;
    g.spacc=0;
    break;
case Tag_address:
    g.spacc=0;
    g.linebrk=1;
    g.state->font=ROMAN;
    g.state->size=NORMAL;
    g.state->margin=300;
    g.state->indent=50;
    break;
case Tag_b:
case Tag_strong:
    g.state->font=BOLD;
    break;
case Tag_s:
case Tag_strike:
case Tag_del:
    g.state->strike=1;
    break;
case Tag_sub:
    g.state->sub++;
    break;
case Tag_sup:
    g.state->sub--;
    break;
case Tag_blockquot:
    g.spacc=0;
    g.linebrk=1;
    g.state->margin+=50;
    g.state->indent=20;
    break;
case Tag_body:
    break;
case Tag_head:
    g.state->font=ROMAN;
    g.state->size=NORMAL;
    g.state->margin=0;
    g.state->indent=20;
```

```
    g.spacc=0;
    break;
case Tag_div:
    g.spacc=0;
    break;
case Tag_br:
case Tag_wbr:
    g.spacc=0;
    g.linebrk=1;
    break;
case Tag_span:
case Tag_center:
    /* more to come */
    break;
case Tag_cite:
case Tag_acronym:
    g.state->font=ITALIC;
    g.state->size=NORMAL;
    break;
case Tag_code:
case Tag_samp:
    g.state->font=CWIDTH;
    g.state->size=NORMAL;
    break;
case Tag_dd:
    g.linebrk=1;
    g.state->indent=0;
    g.state->font=ROMAN;
    g.spacc=0;
    break;
case Tag_dfn:
    htmlerror(g.name, g.lineno, "<dfn> deprecated");
case Tag_abbr:
    g.state->font=BOLD;
    g.state->size=NORMAL;
    break;
case Tag_dl:
    g.state->font=BOLD;
    g.state->size=NORMAL;
    g.state->margin+=40;
    g.spacc=0;
    break;
case Tag_dt:
    g.para=1;
    g.state->indent=-40;
    g.state->font=BOLD;
    g.spacc=0;
    break;
case Tag_figcaption:
    g.linebrk=1;
    break;
case Tag_font:
    /* more to come */
    break;
case Tag_u:
    htmlerror(g.name, g.lineno, "<u> deprecated");
case Tag_ins:
case Tag_em:
case Tag_i:
case Tag_var:
```

```
    g.state->font=ITALIC;
    break;
case Tag_h1:
    g.linebrk=1;
    g.state->font=BOLD;
    g.state->size=ENORMOUS;
    g.state->margin+=100;
    g.spacc=0;
    break;
case Tag_h2:
    pl_linespace(&g);
    g.state->font=BOLD;
    g.state->size=ENORMOUS;
    g.spacc=0;
    break;
case Tag_h3:
    g.linebrk=1;
    pl_linespace(&g);
    g.state->font=ITALIC;
    g.state->size=ENORMOUS;
    g.state->margin+=20;
    g.spacc=0;
    break;
case Tag_h4:
    pl_linespace(&g);
    g.state->font=BOLD;
    g.state->size=LARGE;
    g.state->margin+=10;
    g.spacc=0;
    break;
case Tag_h5:
    pl_linespace(&g);
    g.state->font=ITALIC;
    g.state->size=LARGE;
    g.state->margin+=10;
    g.spacc=0;
    break;
case Tag_h6:
    pl_linespace(&g);
    g.state->font=BOLD;
    g.state->size=LARGE;
    g.spacc=0;
    break;
case Tag_hr:
    g.spacc=0;
    plrtbitmap(&g.dst->text, 1000000, g.state->margin, /*0,*/ hrule, 0, 0);
    break;
case Tag_key:
    htmlerror(g.name, g.lineno, "<key> deprecated");
case Tag_kbd:
    g.state->font=CWIDTH;
    break;
case Tag_dir:
case Tag_menu:
case Tag_ol:
case Tag_ul:
    g.state->number=0;
    g.linebrk=1;
    g.state->margin+=25;
    g.state->indent=-25;
```

```
    g.spacc=0;
    break;
case Tag_li:
    g.spacc=0;
    switch(g.state->tag){
    default:
        htmlerror(g.name, g.lineno, "can't have <li> in <%s>",
            tag[g.state->tag].name);
    case Tag_dir:       /* supposed to be multi-columns, can't do! */
    case Tag_menu:
        g.linebrk=1;
        break;
    case Tag_ol:
        g.para=1;
        snprint(buf, sizeof(buf), "%2d  ", ++g.state->number);
        pl_htmloutput(&g, 0, buf, 0);
        break;
    case Tag_ul:
        g.para=0;
        g.linebrk=0;
        g.spacc=-1;
        plrtbitmap(&g.dst->text, 100000,
            g.state->margin+g.state->indent, /*0,*/ bullet, 0, 0);
        break;
    }
    break;
case Tag_p:
    pl_linespace(&g);
    g.linebrk=1;
    g.spacc=0;
    break;
case Tag_listing:
case Tag_xmp:
    htmlerror(g.name, g.lineno, "<%s> deprecated", tag[g.tag].name);
case Tag_pre:
    g.state->indent=0;
    g.state->pre=1;
    g.state->font=CWIDTH;
    g.state->size=NORMAL;
    pl_linespace(&g);
    break;
case Tag_tt:
    g.state->font=CWIDTH;
    g.state->size=NORMAL;
    break;
case Tag_title:
    g.text=dst->title+strlen(dst->title);
    g.tp=g.text;
    g.etext=dst->title+NTITLE-1;
    break;
case Tag_form:
case Tag_input:
case Tag_button:
case Tag_select:
case Tag_option:
case Tag_textarea:
case Tag_isindex:
    rdform(&g);
    break;
case Tag_script:
```

```
    case Tag_style:
        g.state->isscript=1;
        break;
    }
    break;

case ENDTAG:
    /*
     * If the end tag doesn't match the top, we try to uncover a match
     * on the stack.
     */
    if(g.state->tag!=g.tag){
        tagerr=0;
        for(sp=g.state;sp!=g.stack;--sp){
            if(sp->tag==g.tag)
                break;
            if(tag[g.state->tag].action!=OPTEND) tagerr++;
        }
        if(sp==g.stack){
            if(tagerr)
                htmlerror(g.name, g.lineno,
                    "end tag mismatch <%s>...</%s>, ignored",
                    tag[g.state->tag].name, tag[g.tag].name);
        }
        else{
            if(tagerr)
                htmlerror(g.name, g.lineno,
                    "end tag mismatch <%s>...</%s>, "
                    "intervening tags popped",
                    tag[g.state->tag].name, tag[g.tag].name);

            for(--sp; g.state!=sp; --g.state)
                pl_popstate(g.state);
        }
    }
    else if(g.state==g.stack)
        htmlerror(g.name, g.lineno, "end tag </%s> at stack bottom",
            tag[g.tag].name);
    else
        pl_popstate(g.state--);
    switch(g.tag){
    case Tag_select:
    case Tag_form:
    case Tag_textarea:
        endform(&g);
        break;
    case Tag_h1:
    case Tag_h2:
    case Tag_h3:
    case Tag_h4:
        pl_linespace(&g);
        break;
    case Tag_div:
    case Tag_address:
    case Tag_blockquot:
    case Tag_body:
    case Tag_dir:
    case Tag_dl:
    case Tag_dt:
    case Tag_h5:
```

```
            case Tag_h6:
            case Tag_listing:
            case Tag_menu:
            case Tag_ol:
            case Tag_title:
            case Tag_ul:
            case Tag_xmp:
            case Tag_table:
                g.linebrk=1;
                break;
            case Tag_article:
            case Tag_pre:
                pl_linespace(&g);
                break;
            }
            break;
        case TEXT:
            if(g.state->isscript)
                continue;
            if(g.state->link==0 && (str = linkify(g.token))){
                g.state->link=str;
                pl_htmloutput(&g, g.nsp, g.token, 0);
                free(g.state->link);
                g.state->link=0;
            } else
                pl_htmloutput(&g, g.nsp, g.token, 0);
            break;
        case EOF:
            for(;g.state!=g.stack;--g.state){
                if(tag[g.state->tag].action!=OPTEND)
                    htmlerror(g.name, g.lineno,
                        "missing </%s> at EOF", tag[g.state->tag].name);
                pl_popstate(g.state);
            }
            pl_popstate(g.state);
            *g.tp='\0';
            if (!killimgs)
                getpix(dst->text, dst);
            finish(dst);
            return;
        }
    }
```

⟨mothra/rdhtml.c 206⟩≡

```
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <event.h>
#include <panel.h>
#include "mothra.h"
#include "html.h"
#include "rtext.h"

typedef struct Fontdata Fontdata;
```

⟨*global* `fontlist` *(mothra)* 184⟩

⟨*global* `links` *(mothra)* 185a⟩

⟨*function* `pl_whichfont` *(mothra)* 185b⟩

⟨*function* `getfonts`*(mothra)* 186a⟩
⟨*function* `pl_pushstate`*(mothra)* 186b⟩
⟨*function* `pl_popstate`*(mothra)* 186c⟩

⟨*function* `pl_linespace`*(mothra)* 186d⟩

⟨*function* `strtolength`*(mothra)* 186e⟩

⟨*function* `pl_htmloutput`*(mothra)* 187⟩

⟨*function* `pl_bread`*(mothra)* 188a⟩
⟨*function* `pl_readc`*(mothra)* 188b⟩
⟨*function* `pl_putback`*(mothra)* 189a⟩
⟨*function* `pl_nextc`*(mothra)* 189b⟩

⟨*function* `unquot`*(mothra)* 189c⟩
⟨*function* `alnumchar`*(mothra)* 190a⟩
⟨*function* `entchar`*(mothra)* 190b⟩

⟨*function* `linkify`*(mothra)* 190c⟩

⟨*function* `pl_rmentities`*(mothra)* 190d⟩
⟨*function* `pl_whitespace`*(mothra)* 191a⟩
⟨*function* `pl_word`*(mothra)* 191b⟩
⟨*function* `pl_quote`*(mothra)* 192a⟩
⟨*function* `pl_dnl`*(mothra)* 192b⟩
⟨*function* `pl_tagparse`*(mothra)* 192c⟩
⟨*function* `pl_getcomment`*(mothra)* 193⟩

⟨*function* `lrunetochar`*(mothra)* 194a⟩

⟨*function* `pl_getscript`*(mothra)* 194b⟩

⟨*function* `pl_gettag`*(mothra)* 194c⟩
⟨*function* `pl_gettoken`*(mothra)* 195⟩
⟨*function* `pl_getattr`*(mothra)* 196a⟩
⟨*function* `pl_hasattr`*(mothra)* 196b⟩
⟨*function* `plaintext`*(mothra)* 196c⟩
⟨*function* `plrdplain`*(mothra)* 197a⟩
⟨*function* `plrdhtml`*(mothra)* 197b⟩

mothra/snoop.c

⟨*function* `filetype`*(mothra)* 207⟩≡ (209c)

```
int
filetype(int fd, char *typ, int ntyp)
{
    int ifd[2], ofd[2], xfd[2], n;
    char *argv[3], buf[4096];

    typ[0] = 0;
    if((n = readn(fd, buf, sizeof(buf))) < 0)
        return -1;
    if(n == 0)
        return 0;
    if(pipe(ifd) < 0)
        return -1;
    if(pipe(ofd) < 0){
```

```
Err1:
        close(ifd[0]);
        close(ifd[1]);
        return -1;
    }
    switch(rfork(RFFDG|RFPROC|RFNOWAIT)){
    case -1:
        close(ofd[0]);
        close(ofd[1]);
        goto Err1;
    case 0:
        dup(ifd[1], 0);
        dup(ofd[1], 1);

        close(ifd[1]);
        close(ifd[0]);
        close(ofd[1]);
        close(ofd[0]);
        close(fd);

        argv[0] = "file";
        argv[1] = "-m";
        argv[2] = 0;
        exec("/bin/file", argv);
    }
    close(ifd[1]);
    close(ofd[1]);

    if(rfork(RFFDG|RFPROC|RFNOWAIT) == 0){
        close(fd);
        close(ofd[0]);
        write(ifd[0], buf, n);
        exits(nil);
    }
    close(ifd[0]);

    if(pipe(xfd) < 0){
        close(ofd[0]);
        return -1;
    }
    switch(rfork(RFFDG|RFPROC|RFNOWAIT)){
    case -1:
        break;
    case 0:
        close(ofd[0]);
        close(xfd[0]);
        do {
            if(write(xfd[1], buf, n) != n)
                break;
        } while((n = read(fd, buf, sizeof(buf))) > 0);
        exits(nil);
    default:
        dup(xfd[0], fd);
    }
    close(xfd[0]);
    close(xfd[1]);

    if((n = readn(ofd[0], typ, ntyp-1)) < 0)
        n = 0;
    close(ofd[0]);
```

```
    while(n > 0 && typ[n-1] == '\n')
        n--;
    typ[n] = 0;
    return 0;
}
```

⟨*function* `mimetotype`*(mothra)* 209a⟩≡ (209c)

```
int
mimetotype(char *mime)
{
    int i;
    static struct {
        char    *typ;
        int     val;
    } tab[] = {
    "text/plain",                       PLAIN,
    "text/html",                        HTML,

    "image/jpeg",                       JPEG,
    "image/gif",                        GIF,
    "image/png",                        PNG,
    "image/bmp",                        BMP,
    "image/x-icon",                     ICO,

    "application/pdf",          PAGE,
    "application/postscript",   PAGE,
    "application/ghostscript",  PAGE,
    "application/troff",                PAGE,

    "image/",                   PAGE,
    "text/",                    PLAIN,
    "message/rfc822",           PLAIN,
    };

    for(i=0; i<nelem(tab); i++)
        if(strncmp(mime, tab[i].typ, strlen(tab[i].typ)) == 0)
            return tab[i].val;

    return -1;
}
```

⟨*function* `snooptype`*(mothra)* 209b⟩≡ (209c)

```
int
snooptype(int fd)
{
    char buf[128];

    if(filetype(fd, buf, sizeof(buf)) < 0)
        return -1;

    return mimetotype(buf);
}
```

⟨`mothra/snoop.c` 209c⟩≡

```
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <event.h>
#include <panel.h>
#include <ctype.h>
```

```
#include "mothra.h"
```

⟨*function* `filetype` *(mothra)* 207⟩

⟨*function* `mimetotype` *(mothra)* 209a⟩

⟨*function* `snooptype` *(mothra)* 209b⟩

mothra/url.c

⟨*function* `hexdigit` *(mothra)* 210a⟩≡ (214b)

```
static int
hexdigit(int c)
{
    if(c >= '0' && c <= '9')
        return c-'0';
    if(c >= 'a' && c <= 'f')
        return c-'a'+10;
    if(c >= 'A' && c <= 'F')
        return c-'A'+10;
    return -1;
}
```

⟨*function* `dechex` *(mothra)* 210b⟩≡ (214b)

```
static int
dechex(uchar *out, int lim, char *in, int n)
{
    uchar *start, *end;
    int c;

    start = out;
    end = start + lim;
    while(n-- > 0 && out < end){
        c = *in++;
        if(c == 0)
            break;
        if(c & 0x80)
            return -1;
        if(c == '%'){
            n -= 2;
            if(n < 0 || (c = hexdigit(*in++)) == -1)
                return -1;
            if((c = (c << 4) | hexdigit(*in++)) == -1)
                return -1;
        }
        *out++ = c;
    }
    return out - start;
}
```

⟨*function* `dataget` *(mothra)* 210c⟩≡ (214b)

```
static int
dataget(Url *url)
{
    int (*decfun)(uchar *, int, char *, int) = dechex;
    char *s, *p;
    int fd, n, m;
```

```
    s = url->reltext;
    if(cistrncmp(s, "data:", 5) != 0)
        return -1;
    s += 5;
    if((p = strchr(s, ',')) != nil){
        *p = 0;
        if(strstr(s, "base64") != nil)
            decfun = dec64;
        *p = ',';
        s = p+1;
    } else
        s = strchr(s, 0);
    n = strlen(s);
    m = n+64;
    p = malloc(m);
    strcpy(p, "/tmp/duXXXXXXXXXX");
    if((fd = create(mktemp(p), ORDWR|ORCLOSE, 0600)) < 0){
        free(p);
        return -1;
    }
    if((m = (*decfun)((uchar*)p, m, s, n)) < 0 || write(fd, p, m) != m){
        free(p);
        close(fd);
        return -1;
    }
    free(p);
    seek(fd, 0, 0);
    return fd;
}
```

⟨*function* `fileget` *(mothra)* 211⟩≡ (214b)

```
static int
fileget(Url *url)
{
    char *rel, *base, *x;

    rel = base = nil;
    if(cistrncmp(url->basename, "file:", 5) == 0)
        base = url->basename+5;
    if(cistrncmp(url->reltext, "file:", 5) == 0)
        rel = url->reltext+5;
    if(rel == nil && base == nil)
        return -1;
    if(rel == nil)
        rel = url->reltext;
    if(base && base[0] == '/' && rel[0] != '/'){
        if(x = strrchr(base, '/'))
            *x = 0;
        snprint(url->fullname, sizeof(url->fullname), "%s/%s", base, rel);
        if(x)   *x = '/';
    }else
        snprint(url->fullname, sizeof(url->fullname), "%s", rel);
    url->tag[0] = 0;
    if(x = strrchr(url->fullname, '#')){
        *x++ = 0;
        nstrcpy(url->tag, x, sizeof(url->tag));
    }
    base = cleanname(url->fullname);
    x = base + strlen(base)+1;
    if((x - base) > sizeof(url->fullname)-5)
```

```
        return -1;
    memmove(url->fullname+5, base, x - base);
    memmove(url->fullname, "file:", 5);
    return open(url->fullname+5, OREAD);
}
```

⟨*function* `webclone` *(mothra)* 212a⟩≡ (214b)

```
static fdt
webclone(Url *url, char *buf, int nbuf)
{
    int n, conn;
    fdt fd;

    snprint(buf, nbuf, "%s/clone", mtpt);
    if((fd = open(buf, ORDWR)) < 0)
        return -1;
    if((n = read(fd, buf, nbuf-1)) <= 0){
        close(fd);
        return -1;
    }
    buf[n] = 0;
    conn = atoi(buf);
    if(url && url->reltext[0]){
        if(url->basename[0]){
            n = snprint(buf, nbuf, "baseurl %s", url->basename);
            write(fd, buf, n);
        }
        n = snprint(buf, nbuf, "url %s", url->reltext);
        if(write(fd, buf, n) < 0){
            close(fd);
            return -1;
        }
    }
    snprint(buf, nbuf, "%s/%d", mtpt, conn);
    return fd;
}
```

⟨*function* `readstr` *(mothra)* 212b⟩≡ (214b)

```
static int
readstr(char *path, char *buf, int nbuf){
    int n;
    fdt fd;

    n = 0;
    if((fd = open(path, OREAD)) >= 0){
        if((n = read(fd, buf, nbuf-1)) < 0)
            n = 0;
        close(fd);
    }
    buf[n] = 0;
    return n;
}
```

⟨*function* `urlpost` *(mothra)* 212c⟩≡ (214b)

```
int
urlpost(Url *url, char *ctype)
{
    char buf[1024];
    int n, fd;
```

```
    if((fd = webclone(url, buf, sizeof(buf))) < 0)
        return -1;
    if(ctype && *ctype)
        fprint(fd, "contenttype %s", ctype);
    n = strlen(buf);
    snprint(buf+n, sizeof(buf)-n, "/postbody");
    n = open(buf, OWRITE);
    close(fd);
    return n;
}
```

⟨*function* `urlget` *(mothra)* 213⟩≡ (214b)

```
int
urlget(Url *url, int body)
{
    char buf[1024];
    int n, fd;

    if(body < 0){
        if((fd = dataget(url)) >= 0)
            return fd;
        if((fd = fileget(url)) >= 0)
            return fd;
        if((fd = webclone(url, buf, sizeof(buf))) < 0)
            return -1;
    }else{
        char *x;

        if(fd2path(body, buf, sizeof(buf))){
            close(body);
            return -1;
        }
        if(x = strrchr(buf, '/'))
            *x = 0;
        fd = open(buf, OREAD);
        close(body);
    }
    n = strlen(buf);
    snprint(buf+n, sizeof(buf)-n, "/body");
    body = open(buf, OREAD);
    if(body < 0){
        snprint(buf+n, sizeof(buf)-n, "/errorbody");
        body = open(buf, OREAD);
    }
    close(fd);
    fd = body;
    if(fd < 0)
        return -1;

    snprint(buf+n, sizeof(buf)-n, "/parsed/url");
    readstr(buf, url->fullname, sizeof(url->fullname));

    snprint(buf+n, sizeof(buf)-n, "/parsed/fragment");
    readstr(buf, url->tag, sizeof(url->tag));

    snprint(buf+n, sizeof(buf)-n, "/contenttype");
    readstr(buf, url->contenttype, sizeof(url->contenttype));

    snprint(buf+n, sizeof(buf)-n, "/contentencoding");
    readstr(buf, buf, sizeof(buf));
```

```
    if(!cistrcmp(buf, "compress"))
        fd = pipeline(fd, "exec uncompress");
    else if(!cistrcmp(buf, "gzip"))
        fd = pipeline(fd, "exec gunzip");
    else if(!cistrcmp(buf, "bzip2"))
        fd = pipeline(fd, "exec bunzip2");

    return fd;
}
```

⟨*function* `urlresolve` *(mothra)* 214a⟩≡ (214b)

```
errorneg1
urlresolve(Url *url)
{
    char buf[1024];
    int n, fd;

    if((fd = webclone(url, buf, sizeof(buf))) < 0)
        return -1;
    n = strlen(buf);
    snprint(buf+n, sizeof(buf)-n, "/parsed/url");
    readstr(buf, url->fullname, sizeof(url->fullname));
    snprint(buf+n, sizeof(buf)-n, "/parsed/fragment");
    readstr(buf, url->tag, sizeof(url->tag));
    close(fd);
    return 0;
}
```

⟨`mothra/url.c` 214b⟩≡

```
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <event.h>
#include <panel.h>

#include "mothra.h"
```

⟨*function* `hexdigit` *(mothra)* 210a⟩

⟨*function* `dechex` *(mothra)* 210b⟩

⟨*function* `dataget` *(mothra)* 210c⟩

⟨*function* `fileget` *(mothra)* 211⟩

```
char *mtpt="/mnt/web";
```

⟨*function* `webclone` *(mothra)* 212a⟩

⟨*function* `readstr` *(mothra)* 212b⟩

⟨*function* `urlpost` *(mothra)* 212c⟩

⟨*function* `urlget` *(mothra)* 213⟩

⟨*function* `urlresolve` *(mothra)* 214a⟩

mothra/version.c

⟨mothra/version.c 215a⟩≡
```
char version[]="Apr-21-15:48:03-CES-2026";
```

E.3 misc/

misc/webcookies.c

⟨*struct* Cookie*(webcookies.c)* 215b⟩≡ (236)
```
struct Cookie
{
    /* external info */
    char*       name;
    char*       value;
    char*       dom;            /* starts with . */
    char*       path;
    char*       version;
    char*       comment;        /* optional, may be nil */

    uint        expire;         /* time of expiration: ~0 means when webcookies dies */
    int secure;
    int explicitdom;    /* dom was explicitly set */
    int explicitpath;   /* path was explicitly set */
    int netscapestyle;

    /* internal info */
    int deleted;
    int mark;
    int ondisk;
};
```

⟨*struct* Jar*(webcookies.c)* 215c⟩≡ (236)
```
struct Jar
{
    Cookie      *c;
    int nc;
    int mc;

    Qid qid;
    int dirty;
    char        *file;
    char        *lockfile;
};
```

⟨*global* stab*(webcookies.c)* 215d⟩≡ (236)
```
struct {
    char        *s;
    int offset;
    int ishttp;
} stab[] = {
    "domain",           offsetof(Cookie, dom),          1,
    "path",                     offsetof(Cookie, path),         1,
    "name",                     offsetof(Cookie, name),         0,
    "value",            offsetof(Cookie, value),        0,
    "comment",          offsetof(Cookie, comment),      1,
    "version",          offsetof(Cookie, version),      1,
};
```

⟨*global* `itab` *(webcookies.c)* 216a⟩≡ (236)

```
struct {
    char *s;
    int offset;
} itab[] = {
    "expire",           offsetof(Cookie, expire),
    "secure",           offsetof(Cookie, secure),
    "explicitdomain",   offsetof(Cookie, explicitdom),
    "explicitpath",             offsetof(Cookie, explicitpath),
    "netscapestyle",    offsetof(Cookie, netscapestyle),
};
```

⟨*global* `fs` *(webcookies.c)* 216b⟩≡ (236)

```
Srv fs =
{
.open=          fsopen,
.read=          fsread,
.write=         fswrite,
.destroyfid=    fsdestroyfid,
.end=           fsend,
};
```

⟨*function* `jarfmt` *(webcookies)* 216c⟩≡ (236)

```
/* HTTP format */
int
jarfmt(Fmt *fmt)
{
    int i;
    Jar *jar;

    jar = va_arg(fmt->args, Jar*);
    if(jar == nil || jar->nc == 0)
        return fmtstrcpy(fmt, "");

    fmtprint(fmt, "Cookie: ");
    if(jar->c[0].version)
        fmtprint(fmt, "$Version=%s; ", jar->c[0].version);
    for(i=0; i<jar->nc; i++)
        fmtprint(fmt, "%s%s=%s", i ? "; ":"", jar->c[i].name, jar->c[i].value);
    fmtprint(fmt, "\r\n");
    return 0;
}
```

⟨*function* `cookiefmt` *(webcookies)* 216d⟩≡ (236)

```
/* individual cookie */
int
cookiefmt(Fmt *fmt)
{
    int j, k, first;
    char *t;
    Cookie *c;

    c = va_arg(fmt->args, Cookie*);

    first = 1;
    for(j=0; j<nelem(stab); j++){
        t = *(char**)((char*)c+stab[j].offset);
        if(t == nil)
            continue;
        if(first)
```

```
            first = 0;
        else
            fmtprint(fmt, " ");
        fmtprint(fmt, "%s=%q", stab[j].s, t);
    }
    for(j=0; j<nelem(itab); j++){
        k = *(int*)((char*)c+itab[j].offset);
        if(k == 0)
            continue;
        if(first)
            first = 0;
        else
            fmtprint(fmt, " ");
        fmtprint(fmt, "%s=%ud", itab[j].s, k);
    }
    return 0;
}
```

⟨*function* cookiecmp*(webcookies)* 217a⟩≡ (236)

```
/*
 * sort cookies:
 *      - alpha by name
 *      - alpha by domain
 *      - longer paths first, then alpha by path (RFC2109 4.3.4)
 */
int
cookiecmp(Cookie *a, Cookie *b)
{
    int i;

    if((i = strcmp(a->name, b->name)) != 0)
        return i;
    if((i = cistrcmp(a->dom, b->dom)) != 0)
        return i;
    if((i = strlen(b->path) - strlen(a->path)) != 0)
        return i;
    if((i = strcmp(a->path, b->path)) != 0)
        return i;
    return 0;
}
```

⟨*function* exactcookiecmp*(webcookies)* 217b⟩≡ (236)

```
int
exactcookiecmp(Cookie *a, Cookie *b)
{
    int i;

    if((i = cookiecmp(a, b)) != 0)
        return i;
    if((i = strcmp(a->value, b->value)) != 0)
        return i;
    if(a->version || b->version){
        if(!a->version)
            return -1;
        if(!b->version)
            return 1;
        if((i = strcmp(a->version, b->version)) != 0)
            return i;
    }
    if(a->comment || b->comment){
```

```
        if(!a->comment)
            return -1;
        if(!b->comment)
            return 1;
        if((i = strcmp(a->comment, b->comment)) != 0)
            return i;
    }
    if((i = b->expire - a->expire) != 0)
        return i;
    if((i = b->secure - a->secure) != 0)
        return i;
    if((i = b->explicitdom - a->explicitdom) != 0)
        return i;
    if((i = b->explicitpath - a->explicitpath) != 0)
        return i;
    if((i = b->netscapestyle - a->netscapestyle) != 0)
        return i;

    return 0;
}
```

⟨*function* `freecookie` *(webcookies)* 218a⟩≡ (236)

```
void
freecookie(Cookie *c)
{
    int i;

    for(i=0; i<nelem(stab); i++)
        free(*(char**)((char*)c+stab[i].offset));
}
```

⟨*function* `copycookie` *(webcookies)* 218b⟩≡ (236)

```
void
copycookie(Cookie *c)
{
    int i;
    char **ps;

    for(i=0; i<nelem(stab); i++){
        ps = (char**)((char*)c+stab[i].offset);
        if(*ps)
            *ps = estrdup9p(*ps);
    }
}
```

⟨*function* `delcookie` *(webcookies)* 218c⟩≡ (236)

```
void
delcookie(Jar *j, Cookie *c)
{
    int i;

    j->dirty = 1;
    i = c - j->c;
    if(i < 0 || i >= j->nc)
        abort();
    c->deleted = 1;
}
```

⟨*function* addcookie*(webcookies)* 219a⟩≡ (236)

```
void
addcookie(Jar *j, Cookie *c)
{
    int i;

    if(!c->name || !c->value || !c->path || !c->dom){
        fprint(2, "not adding incomplete cookie\n");
        return;
    }

    if(debug)
        fprint(2, "add %K\n", c);

    for(i=0; i<j->nc; i++)
        if(cookiecmp(&j->c[i], c) == 0){
            if(debug)
                fprint(2, "cookie %K matches %K\n", &j->c[i], c);
            if(exactcookiecmp(&j->c[i], c) == 0){
                if(debug)
                    fprint(2, "\texactly\n");
                j->c[i].mark = 0;
                return;
            }
            delcookie(j, &j->c[i]);
        }

    j->dirty = 1;
    if(j->nc == j->mc){
        j->mc += 16;
        j->c = erealloc9p(j->c, j->mc*sizeof(Cookie));
    }
    j->c[j->nc] = *c;
    copycookie(&j->c[j->nc]);
    j->nc++;
}
```

⟨*function* purgejar*(webcookies)* 219b⟩≡ (236)

```
void
purgejar(Jar *j)
{
    int i;

    for(i=j->nc-1; i>=0; i--){
        if(!j->c[i].deleted)
            continue;
        freecookie(&j->c[i]);
        --j->nc;
        j->c[i] = j->c[j->nc];
    }
}
```

⟨*function* addtojar*(webcookies)* 219c⟩≡ (236)

```
void
addtojar(Jar *jar, char *line, int ondisk)
{
    Cookie c;
    int i, j, nf, *pint;
    char *f[20], *attr, *val, **pstr;
```

```
    memset(&c, 0, sizeof c);
    c.expire = ~0;
    c.ondisk = ondisk;
    nf = tokenize(line, f, nelem(f));
    for(i=0; i<nf; i++){
        attr = f[i];
        if((val = strchr(attr, '=')) != nil)
            *val++ = '\0';
        else
            val = "";
        /* string attributes */
        for(j=0; j<nelem(stab); j++){
            if(strcmp(stab[j].s, attr) == 0){
                pstr = (char**)((char*)&c+stab[j].offset);
                *pstr = val;
            }
        }
        /* integer attributes */
        for(j=0; j<nelem(itab); j++){
            if(strcmp(itab[j].s, attr) == 0){
                pint = (int*)((char*)&c+itab[j].offset);
                if(val[0]=='\0')
                    *pint = 1;
                else
                    *pint = strtoul(val, 0, 0);
            }
        }
    }
    if(c.name==nil || c.value==nil || c.dom==nil || c.path==nil){
        if(debug)
            fprint(2, "ignoring fractional cookie %K\n", &c);
        return;
    }
    addcookie(jar, &c);
}
```

⟨*function* `newjar` *(webcookies)* 220a⟩≡ (236)

```
Jar*
newjar(void)
{
    Jar *jar;

    jar = emalloc9p(sizeof(Jar));
    return jar;
}
```

⟨*function* `expirejar` *(webcookies)* 220b⟩≡ (236)

```
int
expirejar(Jar *jar, int exiting)
{
    int i, n;
    uint now;

    now = time(0);
    n = 0;
    for(i=0; i<jar->nc; i++){
        if(jar->c[i].expire < now || (exiting && jar->c[i].expire==~0)){
            delcookie(jar, &jar->c[i]);
            n++;
        }
```

```
    }
    return n;
}
```

⟨*function* `syncjar` *(webcookies)* 221⟩≡ (236)

```
int
syncjar(Jar *jar)
{
    int i, fd;
    char *line;
    Dir *d;
    Biobuf *b;
    Qid q;

    if(jar->file==nil)
        return 0;

    memset(&q, 0, sizeof q);
    if((d = dirstat(jar->file)) != nil){
        q = d->qid;
        if(d->qid.path != jar->qid.path || d->qid.vers != jar->qid.vers)
            jar->dirty = 1;
        free(d);
    }

    if(jar->dirty == 0)
        return 0;

    fd = -1;
    for(i=0; i<50; i++){
        if((fd = create(jar->lockfile, OWRITE, DMEXCL|0666)) < 0){
            sleep(100);
            continue;
        }
        break;
    }
    if(fd < 0){
        if(debug)
            fprint(2, "open %s: %r", jar->lockfile);
        werrstr("cannot acquire jar lock: %r");
        return -1;
    }

    for(i=0; i<jar->nc; i++)    /* mark is cleared by addcookie */
        jar->c[i].mark = jar->c[i].ondisk;

    if((b = Bopen(jar->file, OREAD)) == nil){
        if(debug)
            fprint(2, "Bopen %s: %r", jar->file);
        werrstr("cannot read cookie file %s: %r", jar->file);
        close(fd);
        return -1;
    }
    for(; (line = Brdstr(b, '\n', 1)) != nil; free(line)){
        if(*line == '#')
            continue;
        addtojar(jar, line, 1);
    }
    Bterm(b);
```

```
    for(i=0; i<jar->nc; i++)
        if(jar->c[i].mark)
            delcookie(jar, &jar->c[i]);

    purgejar(jar);

    b = Bopen(jar->file, OWRITE);
    if(b == nil){
        if(debug)
            fprint(2, "Bopen write %s: %r", jar->file);
        close(fd);
        return -1;
    }
    Bprint(b, "# webcookies cookie jar\n");
    Bprint(b, "# comments and non-standard fields will be lost\n");
    for(i=0; i<jar->nc; i++){
        if(jar->c[i].expire == ~0)
            continue;
        Bprint(b, "%K\n", &jar->c[i]);
        jar->c[i].ondisk = 1;
    }
    Bterm(b);

    jar->dirty = 0;
    close(fd);
    if((d = dirstat(jar->file)) != nil){
        jar->qid = d->qid;
        free(d);
    }
    return 0;
}
```

⟨*function* `readjar` *(webcookies)* 222⟩≡ (236)

```
Jar*
readjar(char *file)
{
    char *lock, *p;
    Jar *jar;

    jar = newjar();
    lock = emalloc9p(strlen(file)+10);
    strcpy(lock, file);
    if((p = strrchr(lock, '/')) != nil)
        p++;
    else
        p = lock;
    memmove(p+2, p, strlen(p)+1);
    p[0] = 'L';
    p[1] = '.';
    jar->lockfile = lock;
    jar->file = file;
    jar->dirty = 1;

    if(syncjar(jar) < 0){
        free(jar->file);
        free(jar->lockfile);
        free(jar);
        return nil;
    }
    return jar;
```

```
}
```

⟨*function* `closejar` *(webcookies)* 223a⟩≡ (236)

```
void
closejar(Jar *jar)
{
    int i;

    expirejar(jar, 0);
    if(syncjar(jar) < 0)
        fprint(2, "warning: cannot rewrite cookie jar: %r\n");

    for(i=0; i<jar->nc; i++)
        freecookie(&jar->c[i]);

    free(jar->file);
    free(jar);
}
```

⟨*function* `isdomainmatch` *(webcookies)* 223b⟩≡ (236)

```
/*
 * Domain name matching is per RFC2109, section 2:
 *
 * Hosts names can be specified either as an IP address or a FQHN
 * string.  Sometimes we compare one host name with another.  Host A's
 * name domain-matches host B's if
 *
 * * both host names are IP addresses and their host name strings match
 *   exactly; or
 *
 * * both host names are FQDN strings and their host name strings match
 *   exactly; or
 *
 * * A is a FQDN string and has the form NB, where N is a non-empty name
 *   string, B has the form .B', and B' is a FQDN string.  (So, x.y.com
 *   domain-matches .y.com but not y.com.)
 *
 * Note that domain-match is not a commutative operation: a.b.c.com
 * domain-matches .c.com, but not the reverse.
 *
 * (This does not verify that IP addresses and FQDN's are well-formed.)
 */
int
isdomainmatch(char *name, char *pattern)
{
    int lname, lpattern;

    if(cistrcmp(name, pattern)==0)
        return 1;

    if(strcmp(ipattr(name), "dom")==0 && pattern[0]=='.'){
        lname = strlen(name);
        lpattern = strlen(pattern);
        if(lname >= lpattern && cistrcmp(name+lname-lpattern, pattern)==0)
            return 1;
    }

    return 0;
}
```

⟨*function* iscookiematch*(webcookies)* 224a⟩≡ (236)

```
/*
 * RFC2109 4.3.4:
 *      - domain must match
 *      - path in cookie must be a prefix of request path
 *      - cookie must not have expired
 */
int
iscookiematch(Cookie *c, char *dom, char *path, uint now)
{
    return isdomainmatch(dom, c->dom)
        && strncmp(c->path, path, strlen(c->path))==0
        && c->expire >= now;
}
```

⟨*function* cookiesearch*(webcookies)* 224b⟩≡ (236)

```
/*
 * Produce a subjar of matching cookies.
 * Secure cookies are only included if secure is set.
 */
Jar*
cookiesearch(Jar *jar, char *dom, char *path, int issecure)
{
    int i;
    Jar *j;
    uint now;

    now = time(0);
    j = newjar();
    for(i=0; i<jar->nc; i++)
        if((issecure || !jar->c[i].secure) && iscookiematch(&jar->c[i], dom, path, now))
            addcookie(j, &jar->c[i]);
    if(j->nc == 0){
        closejar(j);
        werrstr("no cookies found");
        return nil;
    }
    qsort(j->c, j->nc, sizeof(j->c[0]), (int(*)(const void*, const void*))cookiecmp);
    return j;
}
```

⟨*function* isbadcookie*(webcookies)* 224c⟩≡ (236)

```
/*
 * RFC2109 4.3.2 security checks
 */
char*
isbadcookie(Cookie *c, char *dom, char *path)
{
    if(strncmp(c->path, path, strlen(c->path)) != 0)
        return "cookie path is not a prefix of the request path";

    if(c->explicitdom && c->dom[0] != '.')
        return "cookie domain doesn't start with dot";

    if(memchr(c->dom+1, '.', strlen(c->dom)-1-1) == nil)
        return "cookie domain doesn't have embedded dots";

    if(!isdomainmatch(dom, c->dom))
        return "request host does not match cookie domain";
```

```
    if(strcmp(ipattr(dom), "dom")==0
    && memchr(dom, '.', strlen(dom)-strlen(c->dom)) != nil)
        return "request host contains dots before cookie domain";

    return 0;
}
```

⟨*function* `isleap`*(webcookies)* 225a⟩≡ (236)

```
/*
 * Sunday, 25-Jan-2002 12:24:36 GMT
 * Sunday, 25 Jan 2002 12:24:36 GMT
 * Sun, 25 Jan 02 12:24:36 GMT
 */
int
isleap(int year)
{
    return year%4==0 && (year%100!=0 || year%400==0);
}
```

⟨*function* `strtotime`*(webcookies)* 225b⟩≡ (236)

```
uint
strtotime(char *s)
{
    char *os;
    int i;
    Tm tm;

    static int mday[2][12] = {
        31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,
        31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,
    };
    static char *wday[] = {
        "Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday",
    };
    static char *mon[] = {
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",
    };

    os = s;
    /* Sunday, */
    for(i=0; i<nelem(wday); i++){
        if(cistrncmp(s, wday[i], strlen(wday[i])) == 0){
            s += strlen(wday[i]);
            break;
        }
        if(cistrncmp(s, wday[i], 3) == 0){
            s += 3;
            break;
        }
    }
    if(i==nelem(wday)){
        if(debug)
            fprint(2, "bad wday (%s)\n", os);
        return -1;
    }
    if(*s++ != ',' || *s++ != ' '){
        if(debug)
            fprint(2, "bad wday separator (%s)\n", os);
```

```
    return -1;
}

/* 25- */
if(!isdigit(s[0]) || !isdigit(s[1]) || (s[2]!='-' && s[2]!=' ')){
    if(debug)
        fprint(2, "bad day of month (%s)\n", os);
    return -1;
}
tm.mday = strtol(s, 0, 10);
s += 3;

/* Jan- */
for(i=0; i<nelem(mon); i++)
    if(cistrncmp(s, mon[i], 3) == 0){
        tm.mon = i;
        s += 3;
        break;
    }
if(i==nelem(mon)){
    if(debug)
        fprint(2, "bad month (%s)\n", os);
    return -1;
}
if(s[0] != '-' && s[0] != ' '){
    if(debug)
        fprint(2, "bad month separator (%s)\n", os);
    return -1;
}
s++;

/* 2002 */
if(!isdigit(s[0]) || !isdigit(s[1])){
    if(debug)
        fprint(2, "bad year (%s)\n", os);
    return -1;
}
tm.year = strtol(s, 0, 10);
s += 2;
if(isdigit(s[0]) && isdigit(s[1]))
    s += 2;
else{
    if(tm.year <= 68)
        tm.year += 2000;
    else
        tm.year += 1900;
}
if(tm.mday==0 || tm.mday > mday[isleap(tm.year)][tm.mon]){
    if(debug)
        fprint(2, "invalid day of month (%s)\n", os);
    return -1;
}
tm.year -= 1900;
if(*s++ != ' '){
    if(debug)
        fprint(2, "bad year separator (%s)\n", os);
    return -1;
}

if(!isdigit(s[0]) || !isdigit(s[1]) || s[2]!=':'
```

```
    || !isdigit(s[3]) || !isdigit(s[4]) || s[5]!=':'
    || !isdigit(s[6]) || !isdigit(s[7]) || s[8]!=' '){
        if(debug)
            fprint(2, "bad time (%s)\n", os);
        return -1;
    }

    tm.hour = atoi(s);
    tm.min = atoi(s+3);
    tm.sec = atoi(s+6);
    if(tm.hour >= 24 || tm.min >= 60 || tm.sec >= 60){
        if(debug)
            fprint(2, "invalid time (%s)\n", os);
        return -1;
    }
    s += 9;

    if(cistrcmp(s, "GMT") != 0){
        if(debug)
            fprint(2, "time zone not GMT (%s)\n", os);
        return -1;
    }
    strcpy(tm.zone, "GMT");
    tm.yday = 0;
    return tm2sec(&tm);
}
```

⟨*function* skipspace*(webcookies)* 227a⟩≡ (236)

```
/*
 * skip linear whitespace.  we're a bit more lenient than RFC2616 2.2.
 */
char*
skipspace(char *s)
{
    while(*s=='\r' || *s=='\n' || *s==' ' || *s=='\t')
        s++;
    return s;
}
```

⟨*function* isnetscape*(webcookies)* 227b⟩≡ (236)

```
/*
 * Try to identify old netscape headers.
 * The old headers:
 *      - didn't allow spaces around the '='
 *      - used an 'Expires' attribute
 *      - had no 'Version' attribute
 *      - had no quotes
 *      - allowed whitespace in values
 *      - apparently separated attr/value pairs with ';' exclusively
 */
int
isnetscape(char *hdr)
{
    char *s;

    for(s=hdr; (s=strchr(s, '=')) != nil; s++){
        if(isspace(s[1]) || (s > hdr && isspace(s[-1])))
            return 0;
        if(s[1]=='"')
            return 0;
```

```
    }
    if(cistrstr(hdr, "version="))
        return 0;
    return 1;
}
```

⟨*function* `parsehttp` *(webcookies)* 228a⟩≡ (236)

```
int
parsehttp(Jar *jar, char *hdr, char *dom, char *path)
{
    static char setcookie[] = "Set-Cookie:";
    char *e, *p, *nextp;
    Cookie c;
    int isns, n;

    isns = isnetscape(hdr);
    n = 0;
    for(p=hdr; p; p=nextp){
        p = skipspace(p);
        if(*p == '\0')
            break;
        nextp = strchr(p, '\n');
        if(nextp != nil)
            *nextp++ = '\0';
        if(debug)
            fprint(2, "?%s\n", p);
        if(cistrncmp(p, setcookie, strlen(setcookie)) != 0)
            continue;
        if(debug)
            fprint(2, "%s\n", p);
        p = skipspace(p+strlen(setcookie));
        for(; *p; p=skipspace(p)){
            if((e = parsecookie(&c, p, &p, isns, dom, path)) != nil){
                if(debug)
                    fprint(2, "parse cookie: %s\n", e);
                break;
            }
            if((e = isbadcookie(&c, dom, path)) != nil){
                if(debug)
                    fprint(2, "reject cookie; %s\n", e);
                continue;
            }
            addcookie(jar, &c);
            n++;
        }
    }
    return n;
}
```

⟨*function* `skipquoted` *(webcookies)* 228b⟩≡ (236)

```
static char*
skipquoted(char *s)
{
    /*
     * Sec 2.2 of RFC2616 defines a "quoted-string" as:
     *
     *  quoted-string  = ( <"> *(qdtext | quoted-pair ) <"> )
     *  qdtext         = <any TEXT except <">>
     *  quoted-pair    = "\" CHAR
     *
```

```
     * TEXT is any octet except CTLs, but including LWS;
     * LWS is [CR LF] 1*(SP | HT);
     * CHARs are ASCII octets 0-127;  (NOTE: we reject 0's)
     * CTLs are octets 0-31 and 127;
     */
    if(*s != '"')
        return s;

    for(s++; 32 <= *s && *s < 127 && *s != '"'; s++)
        if(*s == '\\' && *(s+1) != '\0')
            s++;
    return s;
}
```

⟨*function* skiptoken*(webcookies)* 229a⟩≡ (236)

```
static char*
skiptoken(char *s)
{
    /*
     * Sec 2.2 of RFC2616 defines a "token" as
         *  1*<any CHAR except CTLs or separators>;
     * CHARs are ASCII octets 0-127;
     * CTLs are octets 0-31 and 127;
     * separators are "()<>@,;:\/[]?={}", double-quote, SP (32), and HT (9)
     */
    while(32 <= *s && *s < 127 && strchr("()<>@,;:[]?={}\" \t\\", *s)==nil)
        s++;

    return s;
}
```

⟨*function* skipvalue*(webcookies)* 229b⟩≡ (236)

```
static char*
skipvalue(char *s, int isns)
{
    char *t;

    /*
     * An RFC2109 value is an HTTP token or an HTTP quoted string.
     * Netscape servers ignore the spec and rely on semicolons, apparently.
     */
    if(isns){
        if((t = strchr(s, ';')) == nil)
            t = s+strlen(s);
        return t;
    }
    if(*s == '"')
        return skipquoted(s);
    return skiptoken(s);
}
```

⟨*function* parsecookie*(webcookies)* 229c⟩≡ (236)

```
/*
 * RMID=80b186bb64c03c65fab767f8; expires=Monday, 10-Feb-2003 04:44:39 GMT;
 *      path=/; domain=.nytimes.com
 */
char*
parsecookie(Cookie *c, char *p, char **e, int isns, char *dom, char *path)
{
    int i, done;
```

```
char *t, *u, *attr, *val;

memset(c, 0, sizeof *c);
c->expire = ~0;

/* NAME=VALUE */
t = skiptoken(p);
c->name = p;
p = skipspace(t);
if(*p != '='){
Badname:
    return "malformed cookie: no NAME=VALUE";
}
*t = '\0';
p = skipspace(p+1);
t = skipvalue(p, isns);
if(*t)
    *t++ = '\0';
c->value = p;
p = skipspace(t);
if(c->name[0]=='\0' || c->value[0]=='\0')
    goto Badname;

done = 0;
for(; *p && !done; p=skipspace(p)){
    attr = p;
    t = skiptoken(p);
    u = skipspace(t);
    switch(*u){
    case '\0':
        *t = '\0';
        p = val = u;
        break;
    case ';':
        *t = '\0';
        val = "";
        p = u+1;
        break;
    case '=':
        *t = '\0';
        val = skipspace(u+1);
        p = skipvalue(val, isns);
        if(*p==',')
            done = 1;
        if(*p)
            *p++ = '\0';
        break;
    case ',':
        if(!isns){
            val = "";
            p = u;
            *p++ = '\0';
            done = 1;
            break;
        }
    default:
        if(debug)
            fprint(2, "syntax: %s\n", p);
        return "syntax error";
    }
```

```
        for(i=0; i<nelem(stab); i++)
            if(stab[i].ishttp && cistrcmp(stab[i].s, attr)==0)
                *(char**)((char*)c+stab[i].offset) = val;
        if(cistrcmp(attr, "expires") == 0){
            if(!isns)
                return "non-netscape cookie has Expires tag";
            if(!val[0])
                return "bad expires tag";
            c->expire = strtotime(val);
            if(c->expire == ~0)
                return "cannot parse netscape expires tag";
        }
        if(cistrcmp(attr, "max-age") == 0)
            c->expire = time(0)+atoi(val);
        if(cistrcmp(attr, "secure") == 0)
            c->secure = 1;
    }

    if(c->dom)
        c->explicitdom = 1;
    else
        c->dom = dom;
    if(c->path)
        c->explicitpath = 1;
    else{
        c->path = path;
        if((t = strchr(c->path, '?')) != 0)
            *t = '\0';
        if((t = strrchr(c->path, '/')) != 0)
            *t = '\0';
    }
    c->netscapestyle = isns;
    *e = p;

    return nil;
}
```

⟨*function* `fsopen`*(webcookies)* 231⟩≡ (236)

```
void
fsopen(Req *r)
{
    char *s, *es;
    int i, sz;
    Aux *a;

    switch((uintptr)r->fid->file->aux){
    case Xhttp:
        syncjar(jar);
        a = emalloc9p(sizeof(Aux));
        r->fid->aux = a;
        a->inhttp = emalloc9p(AuxBuf);
        a->outhttp = emalloc9p(AuxBuf);
        break;

    case Xcookies:
        syncjar(jar);
        a = emalloc9p(sizeof(Aux));
        r->fid->aux = a;
        if(r->ifcall.mode&OTRUNC){
            a->ctext = emalloc9p(1);
```

```
            a->ctext[0] = '\0';
        }else{
            sz = 256*jar->nc+1024;      /* BUG should do better */
            a->ctext = emalloc9p(sz);
            a->ctext[0] = '\0';
            s = a->ctext;
            es = s+sz;
            for(i=0; i<jar->nc; i++)
                s = seprint(s, es, "%K\n", &jar->c[i]);
        }
        break;
    }
    respond(r, nil);
}
```

⟨*function* `fsread` *(webcookies)* 232a⟩≡ (236)

```
void
fsread(Req *r)
{
    Aux *a;

    a = r->fid->aux;
    switch((uintptr)r->fid->file->aux){
    case Xhttp:
        if(a->state == NeedUrl){
            respond(r, "must write url before read");
            return;
        }
        r->ifcall.offset = a->rdoff;
        readstr(r, a->outhttp);
        a->rdoff += r->ofcall.count;
        respond(r, nil);
        return;

    case Xcookies:
        readstr(r, a->ctext);
        respond(r, nil);
        return;

    default:
        respond(r, "bug in webcookies");
        return;
    }
}
```

⟨*function* `fswrite` *(webcookies)* 232b⟩≡ (236)

```
void
fswrite(Req *r)
{
    Aux *a;
    int i, sz, hlen, issecure;
    char buf[1024], *p;
    Jar *j;

    a = r->fid->aux;
    switch((uintptr)r->fid->file->aux){
    case Xhttp:
        if(a->state == NeedUrl){
            if(r->ifcall.count >= sizeof buf){
                respond(r, "url too long");
```

```
            return;
        }
        memmove(buf, r->ifcall.data, r->ifcall.count);
        buf[r->ifcall.count] = '\0';
        issecure = 0;
        if(cistrncmp(buf, "http://", 7) == 0)
            hlen = 7;
        else if(cistrncmp(buf, "https://", 8) == 0){
            hlen = 8;
            issecure = 1;
        }else{
            respond(r, "url must begin http:// or https://");
            return;
        }
        if(buf[hlen]=='/'){
            respond(r, "url without host name");
            return;
        }
        p = strchr(buf+hlen, '/');
        if(p == nil)
            a->path = estrdup9p("/");
        else{
            a->path = estrdup9p(p);
            *p = '\0';
        }
        a->dom = estrdup9p(buf+hlen);
        a->state = HaveUrl;
        j = cookiesearch(jar, a->dom, a->path, issecure);
        if(debug){
            fprint(2, "search %s %s got %p\n", a->dom, a->path, j);
            if(j){
                fprint(2, "%d cookies\n", j->nc);
                for(i=0; i<j->nc; i++)
                    fprint(2, "%K\n", &j->c[i]);
            }
        }
        snprint(a->outhttp, AuxBuf, "%J", j);
        if(j)
            closejar(j);
    }else{
        if(strlen(a->inhttp)+r->ifcall.count >= AuxBuf){
            respond(r, "http headers too large");
            return;
        }
        memmove(a->inhttp+strlen(a->inhttp), r->ifcall.data, r->ifcall.count);
    }
    r->ofcall.count = r->ifcall.count;
    respond(r, nil);
    return;

case Xcookies:
    sz = r->ifcall.count+r->ifcall.offset;
    if(sz > strlen(a->ctext)){
        if(sz >= MaxCtext){
            respond(r, "cookie file too large");
            return;
        }
        a->ctext = erealloc9p(a->ctext, sz+1);
        a->ctext[sz] = '\0';
    }
```

```
        memmove(a->ctext+r->ifcall.offset, r->ifcall.data, r->ifcall.count);
        r->ofcall.count = r->ifcall.count;
        respond(r, nil);
        return;

    default:
        respond(r, "bug in webcookies");
        return;
    }
}
```

⟨*function* `fsdestroyfid` *(webcookies)* 234a⟩≡ (236)

```
void
fsdestroyfid(Fid *fid)
{
    char *p, *nextp;
    Aux *a;
    int i;

    a = fid->aux;
    if(a == nil)
        return;
    switch((uintptr)fid->file->aux){
    case Xhttp:
        parsehttp(jar, a->inhttp, a->dom, a->path);
        break;
    case Xcookies:
        for(i=0; i<jar->nc; i++)
            jar->c[i].mark = 1;
        for(p=a->ctext; *p; p=nextp){
            if((nextp = strchr(p, '\n')) != nil)
                *nextp++ = '\0';
            else
                nextp = "";
            addtojar(jar, p, 0);
        }
        for(i=0; i<jar->nc; i++)
            if(jar->c[i].mark)
                delcookie(jar, &jar->c[i]);
        break;
    }
    syncjar(jar);
    free(a->dom);
    free(a->path);
    free(a->inhttp);
    free(a->outhttp);
    free(a->ctext);
    free(a);
}
```

⟨*function* `fsend` *(webcookies)* 234b⟩≡ (236)

```
void
fsend(Srv*)
{
    closejar(jar);
}
```

⟨*function* `usage` *(webcookies)* 234c⟩≡ (236)

```
void
usage(void)
```

```
{
    fprint(2, "usage: webcookies [-f file] [-m mtpt] [-s service]\n");
    exits("usage");
}
```

⟨*function* main*(webcookies)* 235⟩≡ (236)

```
void
main(int argc, char **argv)
{
    char *file, *mtpt, *home, *srv;

    file = nil;
    srv = nil;
    mtpt = "/mnt/webcookies";
    ARGBEGIN{
    case 'D':
        chatty9p++;
        break;
    case 'd':
        debug = 1;
        break;
    case 'f':
        file = EARGF(usage());
        break;
    case 's':
        srv = EARGF(usage());
        break;
    case 'm':
        mtpt = EARGF(usage());
        break;
    default:
        usage();
    }ARGEND

    if(argc != 0)
        usage();

    quotefmtinstall();
    fmtinstall('J', jarfmt);
    fmtinstall('K', cookiefmt);

    if(file == nil){
        home = getenv("home");
        if(home == nil)
            sysfatal("no cookie file specified and no $home");
        file = emalloc9p(strlen(home)+30);
        strcpy(file, home);
        strcat(file, "/lib/webcookies");
    }
    if(access(file, AEXIST) < 0)
        close(create(file, OWRITE, 0666));

    jar = readjar(file);
    if(jar == nil)
        sysfatal("readjar: %r");

    fs.tree = alloctree("cookie", "cookie", DMDIR|0555, nil);
    closefile(createfile(fs.tree->root, "http", "cookie", 0666, (void*)Xhttp));
    closefile(createfile(fs.tree->root, "cookies", "cookie", 0666, (void*)Xcookies));
```

```
    postmountsrv(&fs, srv, mtpt, MREPL);
    exits(nil);
}
```

⟨misc/webcookies.c 236⟩≡

```
/*
 * Cookie file system.  Allows hget and multiple webfs's to collaborate.
 * Conventionally mounted on /mnt/webcookies.
 */

#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ndb.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>
#include <ctype.h>

int debug = 0;

typedef struct Cookie Cookie;
typedef struct Jar Jar;
```

⟨*struct* Cookie*(webcookies.c)* 215b⟩

⟨*struct* Jar*(webcookies.c)* 215c⟩

⟨*global* stab*(webcookies.c)* 215d⟩

⟨*global* itab*(webcookies.c)* 216a⟩

```
#pragma varargck type "J"       Jar*
#pragma varargck type "K"       Cookie*
```

⟨*function* jarfmt*(webcookies)* 216c⟩

⟨*function* cookiefmt*(webcookies)* 216d⟩

⟨*function* cookiecmp*(webcookies)* 217a⟩

⟨*function* exactcookiecmp*(webcookies)* 217b⟩

⟨*function* freecookie*(webcookies)* 218a⟩

⟨*function* copycookie*(webcookies)* 218b⟩

⟨*function* delcookie*(webcookies)* 218c⟩

⟨*function* addcookie*(webcookies)* 219a⟩

⟨*function* purgejar*(webcookies)* 219b⟩

⟨*function* addtojar*(webcookies)* 219c⟩

⟨*function* newjar*(webcookies)* 220a⟩

⟨*function* expirejar*(webcookies)* 220b⟩

⟨*function* syncjar*(webcookies)* 221⟩

⟨*function* `readjar`*(webcookies)* 222⟩

⟨*function* `closejar`*(webcookies)* 223a⟩

⟨*function* `isdomainmatch`*(webcookies)* 223b⟩

⟨*function* `iscookiematch`*(webcookies)* 224a⟩

⟨*function* `cookiesearch`*(webcookies)* 224b⟩

⟨*function* `isbadcookie`*(webcookies)* 224c⟩

⟨*function* `isleap`*(webcookies)* 225a⟩

⟨*function* `strtotime`*(webcookies)* 225b⟩

⟨*function* `skipspace`*(webcookies)* 227a⟩

⟨*function* `isnetscape`*(webcookies)* 227b⟩

```
/*
 * Parse HTTP response headers, adding cookies to jar.
 * Overwrites the headers.  May overwrite path.
 */
char* parsecookie(Cookie*, char*, char**, int, char*, char*);
```
⟨*function* `parsehttp`*(webcookies)* 228a⟩

⟨*function* `skipquoted`*(webcookies)* 228b⟩

⟨*function* `skiptoken`*(webcookies)* 229a⟩

⟨*function* `skipvalue`*(webcookies)* 229b⟩

⟨*function* `parsecookie`*(webcookies)* 229c⟩

```
Jar *jar;

enum
{
    Xhttp = 1,
    Xcookies,

    NeedUrl = 0,
    HaveUrl,
};

typedef struct Aux Aux;
struct Aux
{
    int state;
    char *dom;
    char *path;
    char *inhttp;
    char *outhttp;
    char *ctext;
    int rdoff;
};
enum
{
```

```
    AuxBuf = 4096,
    MaxCtext = 16*1024*1024,
};
```

⟨*function* `fsopen` *(webcookies)* 231⟩

⟨*function* `fsread` *(webcookies)* 232a⟩

⟨*function* `fswrite` *(webcookies)* 232b⟩

⟨*function* `fsdestroyfid` *(webcookies)* 234a⟩

⟨*function* `fsend` *(webcookies)* 234b⟩

⟨*global* `fs` *(webcookies.c)* 216b⟩

⟨*function* `usage` *(webcookies)* 234c⟩

⟨*function* `main` *(webcookies)* 235⟩

misc/uhtml.c

⟨*constant* `IOUNIT` *(uhtml)* 238a⟩≡ (241a)
```
#define IOUNIT (64*1024)
```

⟨*function* `usage` *(uhtml)* 238b⟩≡ (241a)
```
void
usage(void)
{
    fprint(2, "%s [ -p ] [ -c charset ] [ file ]\n", argv0);
    exits("usage");
}
```

⟨*function* `attr` *(uhtml)* 238c⟩≡ (241a)
```
char*
attr(char *s, char *a)
{
    char *e, q;

    if((s = cistrstr(s, a)) == nil)
        return nil;
    s += strlen(a);
    while(*s && strchr(whitespace, *s))
        s++;
    if(*s++ != '=')
        return nil;
    while(*s && strchr(whitespace, *s))
        s++;
    q = 0;
    if(*s == '"' || *s == '\'')
        q = *s++;
    for(e = s; *e; e++){
        if(*e == q)
            break;
        if(isalnum(*e))
            continue;
        if(*e == '-' || *e == '_')
            continue;
        break;
```

```
    }
    if((e - s) > 1)
        return smprint("%.*s", (int)(e - s), s);
    return nil;
}
```

⟨*function* `main` *(uhtml)* 239⟩≡ (241a)

```
void
main(int argc, char *argv[])
{
    int n, q, pfd[2], pflag = 0;
    char *arg[4], *s, *g, *e, *p, *a, t;
    Rune r;

    ARGBEGIN {
    case 'c':
        cset = EARGF(usage());
        break;
    case 'p':
        pflag = 1;
        break;
    default:
        usage();
    } ARGEND;
    if(*argv){
        close(0);
        if(open(*argv, OREAD) != 0)
            sysfatal("open: %r");
    }
    nbuf = 0;
    while(nbuf < sizeof(buf)-1){
        if((n = read(0, buf + nbuf, sizeof(buf)-1-nbuf)) <= 0)
            break;
        nbuf += n;
        buf[nbuf] = 0;
    }

    p = buf;
    if(nbuf >= 3 && memcmp(p, "\xEF\xBB\xBF", 3)==0){
        p += 3;
        nbuf -= 3;
        cset = "utf";
        goto Found;
    }
    if(nbuf >= 2 && memcmp(p, "\xFE\xFF", 2) == 0){
        p += 2;
        nbuf -= 2;
        cset = "unicode-be";
        goto Found;
    }
    if(nbuf >= 2 && memcmp(p, "\xFF\xFE", 2) == 0){
        p += 2;
        nbuf -= 2;
        cset = "unicode-le";
        goto Found;
    }

    s = p;
    do {
        if((s = strchr(s, '<')) == nil)
```

```
            break;
        q = 0;
        g = ++s;
        e = buf+nbuf;
        while(s < e){
            if(*s == '=' && q == 0)
                q = '=';
            else if(*s == '\'' || *s == '"'){
                if(q == '=')
                    q = *s;
                else if(q == *s)
                    q = 0;
            }
            else if(*s == '>' && q != '\'' && q != '"'){
                e = s;
                break;
            }
            else if(q == '=' && strchr(whitespace, *s) == nil)
                q = 0;
            s++;
        }
        t = *e;
        *e = 0;
        if((a = attr(g, "encoding")) != nil || (a = attr(g, "charset")) != nil)
        if(cistrcmp(a, "utf") != 0 && cistrcmp(a, "utf-8") != 0){
            cset = a;
            *e = t;
            break;
        }
        *e = t;
        s = ++e;
    } while(t);

    s = p;
    while(s+UTFmax < p+nbuf){
        s += chartorune(&r, s);
        if(r == Runeerror){
            if(cset == nil)
                cset = "latin1";
            goto Found;
        }
    }
    cset = "utf";

Found:
    if(pflag){
        print("%s\n", cset);
        exits(0);
    }

    if(nbuf == 0){
        write(1, p, 0);
        exits(0);
    }

    if(pipe(pfd) < 0)
        sysfatal("pipe: %r");

    switch(rfork(RFFDG|RFREND|RFPROC)){
    case -1:
```

```
        sysfatal("fork: %r");
    case 0:
        dup(pfd[0], 0);
        close(pfd[0]);
        close(pfd[1]);

        arg[0] = "rc";
        arg[1] = "-c";
        arg[2] = smprint("{tcs -f %s || cat} | tcs -f html", cset);
        arg[3] = nil;
        exec("/bin/rc", arg);
    }

    dup(pfd[1], 1);
    close(pfd[0]);
    close(pfd[1]);

    while(nbuf > 0){
        if(write(1, p, nbuf) != nbuf)
            sysfatal("write: %r");
        p = buf;
        if((nbuf = read(0, p, sizeof(buf)-1)) < 0)
            sysfatal("read: %r");
    }
    close(1);
    waitpid();
    exits(0);
}
```

⟨misc/uhtml.c 241a⟩≡

```
// used by mothra, imported from 9front too

#include <u.h>
#include <libc.h>
#include <ctype.h>

// copied from Graphics.nw
⟨constant IOUNIT(uhtml) 238a⟩

int nbuf;
char buf[IOUNIT+1];
char *cset = nil;
char *whitespace = " \t\r\n";

⟨function usage(uhtml) 238b⟩

⟨function attr(uhtml) 238c⟩

⟨function main(uhtml) 239⟩
```

misc/webfsget.c

⟨function xfer(webfsget) 241b⟩≡ (243a)

```
void
xfer(fdt from, fdt to)
{
    char buf[12*1024];
    int n;
```

```
    while((n = read(from, buf, sizeof buf)) > 0)
        if(write(to, buf, n) < 0)
            sysfatal("write failed: %r");
    if(n < 0)
        sysfatal("read failed: %r");
}
```

⟨*function* usage*(webfsget)* 242a⟩≡ (243a)

```
void
usage(void)
{
    fprint(2, "usage: webfsget [-b baseurl] [-m mtpt] [-p postbody] url\n");
    exits("usage");
}
```

⟨*function* main*(webfsget)* 242b⟩≡ (243a)

```
void
main(int argc, char **argv)
{
    int conn, n;
    fdt ctlfd, fd;
    char buf[128], *base, *mtpt, *post, *url;

    mtpt = "/mnt/web";
    post = nil;
    base = nil;
    ARGBEGIN{
    default:
        usage();
    case 'b':
        base = EARGF(usage());
        break;
    case 'm':
        mtpt = EARGF(usage());
        break;
    case 'p':
        post = EARGF(usage());
        break;
    }ARGEND;
    if (argc != 1)
        usage();

    url = argv[0];

    snprint(buf, sizeof buf, "%s/clone", mtpt);
    if((ctlfd = open(buf, ORDWR)) < 0)
        sysfatal("couldn't open %s: %r", buf);
    if((n = read(ctlfd, buf, sizeof buf-1)) < 0)
        sysfatal("reading clone: %r");
    if(n == 0)
        sysfatal("short read on clone");
    buf[n] = '\0';
    conn = atoi(buf);

    if(base)
        if(fprint(ctlfd, "baseurl %s", base) < 0)
            sysfatal("baseurl ctl write: %r");

    if(fprint(ctlfd, "url %s", url) <= 0)
```

```
        sysfatal("get ctl write: %r");

    if(post){
        snprint(buf, sizeof buf, "%s/%d/postbody", mtpt, conn);
        if((fd = open(buf, OWRITE)) < 0)
            sysfatal("open %s: %r", buf);
        if(write(fd, post, strlen(post)) < 0)
            sysfatal("post write failed: %r");
        close(fd);
    }

    snprint(buf, sizeof buf, "%s/%d/body", mtpt, conn);
    if((fd = open(buf, OREAD)) < 0)
        sysfatal("open %s: %r", buf);

    xfer(fd, STDOUT);
    exits(nil);
}
```

⟨misc/webfsget.c 243a⟩≡

```
// was in src/cmd/webfs/webget.c before and called webget

/* Example of how to use webfs */
#include <u.h>
#include <libc.h>

⟨function xfer(webfsget) 241b⟩

⟨function usage(webfsget) 242a⟩

⟨function main(webfsget) 242b⟩
```

misc/resize.c

⟨function checkimag(resize) 243b⟩≡ (247a)

```
static void
checkimag(char *where)
{
    if(!debug)
        return;
    fprint(2, "resize: poolcheck before %s... ", where);
    poolcheck(imagmem);
    poolcheck(mainmem);
    fprint(2, "ok\n");
}
```

⟨function resample(resize) 243c⟩≡ (247a)

```
static void
resample(Memimage *dst, Rectangle r, Memimage *src, Rectangle sr)
{
    Point sp, dp;
    Point _sp, qp;
    Point ssize, dsize;
    uchar *pdst0, *pdst, *psrc0, *psrc;
    ulong s00, s01, s10, s11;
    int tx, ty, bpp, bpl;

    ssize = subpt(subpt(sr.max, sr.min), Pt(1,1));
    dsize = subpt(subpt(r.max, r.min), Pt(1,1));
```

```
    pdst0 = byteaddr(dst, r.min);
    bpp = src->depth/8;
    bpl = src->width*sizeof(int);

    qp = Pt(0, 0);
    if(dsize.x > 0)
        qp.x = (ssize.x<<12)/dsize.x;
    if(dsize.y > 0)
        qp.y = (ssize.y<<12)/dsize.y;

    _sp.y = sr.min.y<<12;
    for(dp.y=0; dp.y<=dsize.y; dp.y++){
        sp.y = _sp.y>>12;
        ty = _sp.y&0xFFF;
        if(nflag)
            ty = ty << 1 & 0x1000;
        pdst = pdst0;
        sp.x = sr.min.x;
        psrc0 = byteaddr(src, sp);
        _sp.x = 0;
        for(dp.x=0; dp.x<=dsize.x; dp.x++){
            sp.x = _sp.x>>12;
            tx = _sp.x&0xFFF;
            if(nflag)
                tx = tx << 1 & 0x1000;
            psrc = psrc0 + sp.x*bpp;
            s00 = (0x1000-tx)*(0x1000-ty);
            s01 = tx*(0x1000-ty);
            s10 = (0x1000-tx)*ty;
            s11 = tx*ty;
            switch(bpp){
            case 4:
                pdst[3] = (s11*psrc[bpl+bpp+3] +
                       s10*psrc[bpl+3] +
                       s01*psrc[bpp+3] +
                       s00*psrc[3]) >>24;
            case 3:
                pdst[2] = (s11*psrc[bpl+bpp+2] +
                       s10*psrc[bpl+2] +
                       s01*psrc[bpp+2] +
                       s00*psrc[2]) >>24;
                pdst[1] = (s11*psrc[bpl+bpp+1] +
                       s10*psrc[bpl+1] +
                       s01*psrc[bpp+1] +
                       s00*psrc[1]) >>24;
            case 1:
                pdst[0] = (s11*psrc[bpl+bpp] +
                       s10*psrc[bpl] +
                       s01*psrc[bpp] +
                       s00*psrc[0]) >>24;
            }
            pdst += bpp;
            _sp.x += qp.x;
        }
        pdst0 += dst->width*sizeof(int);
        _sp.y += qp.y;
    }
}
```

⟨*function* `getsize` *(resize)* 244⟩≡ (247a)

```
static int
getsize(char *s)
{
    int v;

    v = strtol(s, &s, 10) & ~PERCENT;
    if(*s == '%')
        v |= PERCENT;
    return v;
}
```

⟨*function* usage*(resize)* 245a⟩≡ (247a)

```
void
usage(void)
{
    fprint(2, "Usage: %s [-d] [ -x width ] [ -y height ] [ file ]\n", argv0);
    exits("usage");
}
```

⟨*function* main*(resize)* 245b⟩≡ (247a)

```
void
main(int argc, char **argv)
{
    int fd, xsize, ysize;
    Memimage *im, *nim;
    ulong ochan, tchan;

    xsize = ysize = 0;
    ARGBEGIN{
    case 'a':
        xsize = ysize = getsize(EARGF(usage()));
        break;
    case 'x':
        xsize = getsize(EARGF(usage()));
        break;
    case 'y':
        ysize = getsize(EARGF(usage()));
        break;
    case 'n':
        nflag++;
        break;
    case 'd':
        debug = true;
        break;
    default:
        usage();
    }ARGEND
    fd = 0;
    if(*argv){
        fd = open(*argv, OREAD);
        if(fd < 0)
            sysfatal("open: %r");
    }
    memimageinit();
    if((im = readmemimage(fd)) == nil)
        sysfatal("readmemimage: %r");
    if(xsize & PERCENT)
        xsize = ((xsize & ~PERCENT) * Dx(im->r)) / 100;
    if(ysize & PERCENT)
        ysize = ((ysize & ~PERCENT) * Dy(im->r)) / 100;
```

```
if(xsize || ysize){
    if(ysize == 0)
        ysize = (xsize * Dy(im->r)) / Dx(im->r);
    if(xsize == 0)
        xsize = (ysize * Dx(im->r)) / Dy(im->r);
    ochan = im->chan;
    switch(ochan){
    default:
        for(tchan = ochan; tchan; tchan >>= 8)
            if(TYPE(tchan) == CAlpha){
                tchan = RGBA32;
                break;
            }
        if(tchan == 0)
            tchan = RGB24;
        break;
    case GREY8:
    case RGB24:
    case RGBA32:
    case ARGB32:
    case XRGB32:
        tchan = ochan;
        break;
    case GREY1:
    case GREY2:
    case GREY4:
        tchan = GREY8;
        break;
    }
    checkimag("enter");
    if(tchan != ochan){
        if((nim = allocmemimage(im->r, tchan)) == nil)
            sysfatal("allocimage: %r");
        checkimag("memimagedraw1 (ochan->tchan)");
        memimagedraw(nim, nim->r, im, im->r.min, nil, ZP, S);
        checkimag("freememimage(im) #1");
        freememimage(im);
        im = nim;
    }
    if((nim = allocmemimage(
        Rect(im->r.min.x, im->r.min.y, im->r.min.x+xsize, im->r.min.y+ysize),
        tchan)) == nil)
        sysfatal("allocmemimage: %r");
    checkimag("resample");
    resample(nim, nim->r, im, im->r);
    checkimag("freememimage(im) #2");
    freememimage(im);
    im = nim;
    if(tchan != ochan){
        if((im = allocmemimage(nim->r, ochan)) == nil)
            sysfatal("allocimage: %r");
        checkimag("memimagedraw2 (tchan->ochan)");
        memimagedraw(im, im->r, nim, nim->r.min, nil, ZP, S);
        checkimag("freememimage(nim)");
        freememimage(nim);
    }
}
if(writememimage(1, im) < 0)
    sysfatal("writememimage: %r");
exits(0);
```

```
}
```

⟨misc/resize.c 247a⟩≡

```
// called by mothra via exec
#include <u.h>
#include <libc.h>
#include <pool.h>
#include <draw.h>
#include <memdraw.h>

int nflag;
bool debug; /* claude: -d: poolcheck before each big step, to locate heap corruption */

/*
 * claude: temporary instrumentation to locate the heap corruption that
 * crashes freememimage(nim) at the end of main(). poolcheck walks the
 * imagmem arena and sysfatal()s on the first bad boundary tag. Prints
 * a "CHECK ok" line if clean, so we can see which step introduced the
 * corruption.
 */
extern Pool *imagmem;
extern Pool *mainmem;
⟨function checkimag(resize) 243b⟩

⟨function resample(resize) 243c⟩

enum {
    PERCENT = 0x80000000,
};

⟨function getsize(resize) 244⟩

⟨function usage(resize) 245a⟩

⟨function main(resize) 245b⟩
```

misc/hget.c

⟨struct URL(hget.c) 247b⟩≡ (261b)

```
struct URL
{
    int method;
    char    *host;
    char    *port;
    char    *page;
    char    *etag;
    char    *redirect;
    char    *postbody;
    char    *cred;
    char *rhead;
    long    mtime;
};
```

⟨struct Range(hget.c) 247c⟩≡ (261b)

```
struct Range
{
    long    start;  /* only 2 gig supported, tdb */
    long    end;
};
```

⟨*struct* `Out` *(hget.c)* 248a⟩≡ (261b)

```
struct Out
{
    int fd;
    int offset;             /* notional current offset in output */
    int written;            /* number of bytes successfully transferred to output */
    DigestState *curr;      /* digest state up to offset (if known) */
    DigestState *hiwat;     /* digest state of all bytes written */
};
```

⟨*global* `method` *(hget.c)* 248b⟩≡ (261b)

```
struct {
    char    *name;
    int (*f)(URL*, URL*, Range*, Out*, long);
} method[] = {
    [Http]  { "http",   dohttp },
    [Https] { "https",  dohttp },
    [Ftp]   { "ftp",    doftp },
    [Other] { "_______",    nil },
};
```

⟨*function* `usage` *(hget.c)* 248c⟩≡ (261b)

```
void
usage(void)
{
    fprint(2, "usage: %s [-dhv] [-o outfile] [-p body] [-x netmtpt] [-r header] url\n", argv0);
    exits("usage");
}
```

⟨*function* `main` *(hget.c)* 248d⟩≡ (261b)

```
void
main(int argc, char **argv)
{
    URL u;
    Range r;
    int errs, n;
    ulong mtime;
    Dir *d;
    char postbody[4096], *p, *e, *t, *hpx;
    URL px; // Proxy
    Out out;

    ofile = nil;
    p = postbody;
    e = p + sizeof(postbody);
    r.start = 0;
    r.end = -1;
    mtime = 0;
    memset(&u, 0, sizeof(u));
    memset(&px, 0, sizeof(px));
    hpx = getenv("httpproxy");

    ARGBEGIN {
    case 'o':
        ofile = EARGF(usage());
        break;
    case 'd':
        debug = true;
        break;
    case 'h':
```

```
    headerprint = 1;
    break;
case 'v':
    verbose = 1;
    break;
case 'x':
    net = EARGF(usage());
    break;
case 'r':
    u.rhead = EARGF(usage());
    break;
case 'p':
    t = EARGF(usage());
    if(p != postbody)
        p = seprint(p, e, "&%s", t);
    else
        p = seprint(p, e, "%s", t);
    u.postbody = postbody;

    break;
default:
    usage();
} ARGEND;

if(net != nil){
    if(strlen(net) > sizeof(tcpdir)-5)
        sysfatal("network mount point too long");
    snprint(tcpdir, sizeof(tcpdir), "%s/tcp", net);
} else
    snprint(tcpdir, sizeof(tcpdir), "tcp");

if(argc != 1)
    usage();


out.fd = 1;
out.written = 0;
out.offset = 0;
out.curr = nil;
out.hiwat = nil;
if(ofile != nil){
    d = dirstat(ofile);
    if(d == nil){
        out.fd = create(ofile, OWRITE, 0664);
        if(out.fd < 0)
            sysfatal("creating %s: %r", ofile);
    } else {
        out.fd = open(ofile, OWRITE);
        if(out.fd < 0)
            sysfatal("can't open %s: %r", ofile);
        r.start = d->length;
        mtime = d->mtime;
        free(d);
    }
}

errs = 0;

if(crackurl(&u, argv[0]) < 0)
    sysfatal("%r");
```

```
    if(hpx && crackurl(&px, hpx) < 0)
        sysfatal("%r");

    for(;;){
        setoffset(&out, 0);
        /* transfer data */
        werrstr("");
        n = (*method[u.method].f)(&u, &px, &r, &out, mtime);

        switch(n){
        case Eof:
            exits(0);
            break;
        case Error:
            if(errs++ < 10)
                continue;
            sysfatal("too many errors with no progress %r");
            break;
        case Server:
            sysfatal("server returned: %r");
            break;
        }

        /* forward progress */
        errs = 0;
        r.start += n;
        if(r.start >= r.end)
            break;
    }

    exits(0);
}
```

⟨*function* `crackurl` *(hget.c)* 250⟩≡ (261b)

```
int
crackurl(URL *u, char *s)
{
    char *p;
    int i;

    if(u->page != nil){
        free(u->page);
        u->page = nil;
    }

    /* get type */
    for(p = s; *p; p++){
        if(*p == '/'){
            p = s;
            if(u->method == Other){
                werrstr("missing method");
                return -1;
            }
            if(u->host == nil){
                werrstr("missing host");
                return -1;
            }
            u->page = strdup(p);
            return 0;
        }
```

```
        if(*p == ':' && *(p+1)=='/' && *(p+2)=='/'){
            *p = 0;
            p += 3;
            for(i = 0; i < nelem(method); i++){
                if(cistrcmp(s, method[i].name) == 0){
                    u->method = i;
                    break;
                }
            }
            break;
        }
    }

    if(u->method == Other){
        werrstr("unsupported URL type %s", s);
        return -1;
    }

    /* get system */
    free(u->host);
    s = p;
    p = strchr(s, '/');
    if(p == nil){
        u->host = strdup(s);
        u->page = strdup("/");
    } else {
        u->page = strdup(p);
        *p = 0;
        u->host = strdup(s);
        *p = '/';
    }

    if(p = strchr(u->host, ':')) {
        *p++ = 0;
        u->port = p;
    } else
        u->port = method[u->method].name;

    if(*(u->host) == 0){
        werrstr("bad url, null host");
        return -1;
    }

    return 0;
}
```

⟨*function* dohttp*(hget.c)* 251⟩≡ (261b)

```
int
dohttp(URL *u, URL *px, Range *r, Out *out, long mtime)
{
    int fd, cfd;
    int redirect, auth, loop;
    int n, rv, code;
    long tot, vtime;
    Tm *tm;
    char buf[1024];
    char err[ERRMAX];


    /*  always move back to a previous 512 byte bound because some
```

```
 *  servers can't seem to deal with requests that start at the
 *  end of the file
 */
if(r->start)
    r->start = ((r->start-1)/512)*512;

/* loop for redirects, requires reading both response code and headers */
fd = -1;
for(loop = 0; loop < 32; loop++){
    if(px->host == nil){
        fd = dial(netmkaddr(u->host, tcpdir, u->port), 0, 0, 0);
    } else {
        fd = dial(netmkaddr(px->host, tcpdir, px->port), 0, 0, 0);
    }
    if(fd < 0)
        return Error;

    if(u->method == Https){
        int tfd;
        TLSconn conn;

        memset(&conn, 0, sizeof conn);
        /* claude: SNI is required by most modern HTTPS servers */
        conn.serverName = strdup(u->host);
        tfd = tlsClient(fd, &conn);
        if(tfd < 0){
            fprint(2, "tlsClient: %r\n");
            close(fd);
            free(conn.serverName);
            return Error;
        }
        /* BUG: check cert here? */
        if(conn.cert)
            free(conn.cert);
        free(conn.serverName);
        close(fd);
        fd = tfd;
    }

    /* write request, use range if not start of file */
    if(u->postbody == nil){
        if(px->host == nil){
            dfprint(fd, "GET %s HTTP/1.0\r\n"
                    "Host: %s\r\n"
                    "User-agent: Plan9/hget\r\n"
                    "Cache-Control: no-cache\r\n"
                    "Pragma: no-cache\r\n",
                    u->page, u->host);
        } else {
            dfprint(fd, "GET http://%s%s HTTP/1.0\r\n"
                    "Host: %s\r\n"
                    "User-agent: Plan9/hget\r\n"
                    "Cache-Control: no-cache\r\n"
                    "Pragma: no-cache\r\n",
                    u->host, u->page, u->host);
        }
    } else {
        dfprint(fd, "POST %s HTTP/1.0\r\n"
                "Host: %s\r\n"
                "Content-type: application/x-www-form-urlencoded\r\n"
```

```
            "Content-length: %d\r\n"
            "User-agent: Plan9/hget\r\n",
            u->page, u->host, strlen(u->postbody));
}
if(u->cred)
    dfprint(fd, "Authorization: Basic %s\r\n", u->cred);
if(u->rhead)
    dfprint(fd, "%s\r\n", u->rhead);
if(r->start != 0){
    dfprint(fd, "Range: bytes=%d-\n", r->start);
    if(u->etag != nil){
        dfprint(fd, "If-range: %s\n", u->etag);
    } else {
        tm = gmtime(mtime);
        dfprint(fd, "If-range: %s, %d %s %d %2d:%2.2d:%2.2d GMT\n",
            day[tm->wday], tm->mday, month[tm->mon],
            tm->year+1900, tm->hour, tm->min, tm->sec);
    }
}
if((cfd = open("/mnt/webcookies/http", ORDWR)) >= 0){
    if(fprint(cfd, "http://%s%s", u->host, u->page) > 0){
        while((n = read(cfd, buf, sizeof buf)) > 0){
            if(debug)
                write(STDERR, buf, n);
            write(fd, buf, n);
        }
    }else{
        close(cfd);
        cfd = -1;
    }
}

dfprint(fd, "\r\n", u->host);
if(u->postbody)
    dfprint(fd, "%s", u->postbody);

auth = 0;
redirect = 0;
initibuf();
code = httprcode(fd);
switch(code){
case Error: /* connection timed out */
case Eof:
    close(fd);
    close(cfd);
    return code;

case 200:   /* OK */
case 201:   /* Created */
case 202:   /* Accepted */
    if(ofile == nil && r->start != 0)
        sysfatal("page changed underfoot");
    break;

case 204:   /* No Content */
    sysfatal("No Content");

case 206:   /* Partial Content */
    setoffset(out, r->start);
    break;
```

```
case 301:   /* Moved Permanently */
case 302:   /* Moved Temporarily (actually Found) */
case 303:   /* See Other */
case 307:   /* Temporary Redirect (HTTP/1.1) */
    redirect = 1;
    u->postbody = nil;
    break;

case 304:   /* Not Modified */
    break;

case 400:   /* Bad Request */
    sysfatal("Bad Request");

case 401:   /* Unauthorized */
    if (auth)
        sysfatal("Authentication failed");
    auth = 1;
    break;

case 402:   /* ??? */
    sysfatal("Unauthorized");

case 403:   /* Forbidden */
    sysfatal("Forbidden by server");

case 404:   /* Not Found */
    sysfatal("Not found on server");

case 407:   /* Proxy Authentication */
    sysfatal("Proxy authentication required");

case 500:   /* Internal server error */
    sysfatal("Server choked");

case 501:   /* Not implemented */
    sysfatal("Server can't do it!");

case 502:   /* Bad gateway */
    sysfatal("Bad gateway");

case 503:   /* Service unavailable */
    sysfatal("Service unavailable");

default:
    sysfatal("Unknown response code %d", code);
}

if(u->redirect != nil){
    free(u->redirect);
    u->redirect = nil;
}

rv = httpheaders(fd, cfd, u, r);
close(cfd);
if(rv != 0){
    close(fd);
    return rv;
}
```

```
        if(!redirect && !auth)
            break;

        if (redirect){
            if(u->redirect == nil)
                sysfatal("redirect: no URL");
            if(crackurl(u, u->redirect) < 0)
                sysfatal("redirect: %r");
        }
    }

    /* transfer whatever you get */
    if(ofile != nil && u->mtime != 0){
        note.fd = out->fd;
        note.mtime = u->mtime;
        notify(catch);
    }

    tot = 0;
    vtime = 0;
    for(;;){
        n = readibuf(fd, buf, sizeof(buf));
        if(n <= 0)
            break;
        if(output(out, buf, n) != n)
            break;
        tot += n;
        if(verbose && (vtime != time(0) || r->start == r->end)) {
            vtime = time(0);
            fprint(2, "%ld %ld\n", r->start+tot, r->end);
        }
    }
    notify(nil);
    close(fd);

    if(ofile != nil && u->mtime != 0){
        Dir d;

        rerrstr(err, sizeof err);
        nulldir(&d);
        d.mtime = u->mtime;
        if(dirfwstat(out->fd, &d) < 0)
            fprint(2, "couldn't set mtime: %r\n");
        errstr(err, sizeof err);
    }

    return tot;
}
```

⟨*function* `httprcode` *(hget.c)* 255⟩≡ (261b)

```
/* get the http response code */
int
httprcode(int fd)
{
    int n;
    char *p;
    char buf[256];

    n = readline(fd, buf, sizeof(buf)-1);
```

```
    if(n <= 0)
        return n;
    if(debug)
        fprint(STDERR, "%d <- %s\n", fd, buf);
    p = strchr(buf, ' ');
    if(strncmp(buf, "HTTP/", 5) != 0 || p == nil){
        werrstr("bad response from server");
        return -1;
    }
    buf[n] = 0;
    return atoi(p+1);
}
```

⟨*global* headers *(hget.c)* 256a⟩≡ (261b)

```
struct {
    char *name;
    void (*f)(char*, URL*, Range*);
} headers[] = {
    { "etag:", hhetag },
    { "last-modified:", hhmtime },
    { "content-length:", hhclen },
    { "content-range:", hhcrange },
    { "uri:", hhuri },
    { "location:", hhlocation },
    { "WWW-Authenticate:", hhauth },
};
```

⟨*function* httheaders *(hget.c)* 256b⟩≡ (261b)

```
int
httpheaders(int fd, int cfd, URL *u, Range *r)
{
    char buf[2048];
    char *p;
    int i, n;

    for(;;){
        n = getheader(fd, buf, sizeof(buf));
        if(n <= 0)
            break;
        if(cfd >= 0)
            fprint(cfd, "%s\n", buf);
        for(i = 0; i < nelem(headers); i++){
            n = strlen(headers[i].name);
            if(cistrncmp(buf, headers[i].name, n) == 0){
                /* skip field name and leading white */
                p = buf + n;
                while(*p == ' ' || *p == '\t')
                    p++;

                (*headers[i].f)(p, u, r);
                break;
            }
        }
    }
    return n;
}
```

⟨*function* getheader *(hget.c)* 256c⟩≡ (261b)

```
/*
 *  read a single mime header, collect continuations.
```

```
 *
 *  this routine assumes that there is a blank line twixt
 *  the header and the message body, otherwise bytes will
 *  be lost.
 */
int
getheader(int fd, char *buf, int n)
{
    char *p, *e;
    int i;

    n--;
    p = buf;
    for(e = p + n; ; p += i){
        i = readline(fd, p, e-p);
        if(i < 0)
            return i;

        if(p == buf){
            /* first line */
            if(strchr(buf, ':') == nil)
                break;      /* end of headers */
        } else {
            /* continuation line */
            if(*p != ' ' && *p != '\t'){
                unreadline(p);
                *p = 0;
                break;      /* end of this header */
            }
        }
    }
    if(headerprint)
        print("%s\n", buf);

    if(debug)
        fprint(STDERR, "%d <- %s\n", fd, buf);
    return p-buf;
}
```

⟨*function* doftp*(hget.c)* 257⟩≡ (261b)

```
int
doftp(URL *u, URL *px, Range *r, Out *out, long mtime)
{
    int pid, ctl, data, rv;
    Waitmsg *w;
    char msg[64];
    char conndir[NETPATHLEN];
    char *p;

    /* untested, proxy doesn't work with ftp (I think) */
    if(px->host == nil){
        ctl = dial(netmkaddr(u->host, tcpdir, u->port), 0, conndir, 0);
    } else {
        ctl = dial(netmkaddr(px->host, tcpdir, px->port), 0, conndir, 0);
    }

    if(ctl < 0)
        return Error;
    if(net == nil){
        p = strrchr(conndir, '/');
```

```
    *p = 0;
    snprint(tcpdir, sizeof(tcpdir), conndir);
}

initibuf();

rv = hello(ctl);
if(rv < 0)
    return terminateftp(ctl, rv);

rv = logon(ctl);
if(rv < 0)
    return terminateftp(ctl, rv);

rv = xfertype(ctl, "I");
if(rv < 0)
    return terminateftp(ctl, rv);

/* if file is up to date and the right size, stop */
if(ftprestart(ctl, out, u, r, mtime) > 0){
    close(ctl);
    return Eof;
}

/* first try passive mode, then active */
data = passive(ctl, u);
if(data < 0){
    data = active(ctl, u);
    if(data < 0)
        return Error;
}

/* fork */
switch(pid = rfork(RFPROC|RFFDG|RFMEM)){
case -1:
    close(data);
    return terminateftp(ctl, Error);
case 0:
    ftpxfer(data, out, r);
    close(data);
    _exits(0);
default:
    close(data);
    break;
}

/* wait for reply message */
rv = ftprcode(ctl, msg, sizeof(msg));
close(ctl);

/* wait for process to terminate */
w = nil;
for(;;){
    free(w);
    w = wait();
    if(w == nil)
        return Error;
    if(w->pid == pid){
        if(w->msg[0] == 0){
            free(w);
```

```
                break;
            }
            werrstr("xfer: %s", w->msg);
            free(w);
            return Error;
        }
    }

    switch(rv){
    case Success:
        return Eof;
    case TempFail:
        return Server;
    default:
        return Error;
    }
}
```

⟨*function* `ftpcmt` *(hget.c)* 259a⟩≡ (261b)

```
int
ftpcmd(int ctl, char *fmt, ...)
{
    va_list arg;
    char buf[2*1024], *s;

    va_start(arg, fmt);
    s = vseprint(buf, buf + (sizeof(buf)-4) / sizeof(*buf), fmt, arg);
    va_end(arg);
    if(debug)
        fprint(STDERR, "%d -> %s\n", ctl, buf);
    *s++ = '\r';
    *s++ = '\n';
    if(write(ctl, buf, s - buf) != s - buf)
        return -1;
    return 0;
}
```

⟨*function* `ftprcode` *(hget.c)* 259b⟩≡ (261b)

```
int
ftprcode(int ctl, char *msg, int len)
{
    int rv;
    int i;
    char *p;

    len--;  /* room for terminating null */
    for(;;){
        *msg = 0;
        i = readline(ctl, msg, len);
        if(i < 0)
            break;
        if(debug)
            fprint(STDERR, "%d <- %s\n", ctl, msg);

        /* stop if not a continuation */
        rv = strtol(msg, &p, 10);
        if(rv >= 100 && rv < 600 && p==msg+3 && *p == ' ')
            return rv/100;
    }
    *msg = 0;
```

```
    return -1;
}
```

⟨*function* `hello` *(hget.c)* 260a⟩≡ (261b)

```
int
hello(int ctl)
{
    char msg[1024];

    /* wait for hello from other side */
    if(ftprcode(ctl, msg, sizeof(msg)) != Success){
        werrstr("HELLO: %s", msg);
        return Server;
    }
    return 0;
}
```

⟨*function* `getdec` *(hget.c)* 260b⟩≡ (261b)

```
int
getdec(char *p, int n)
{
    int x = 0;
    int i;

    for(i = 0; i < n; i++)
        x = x*10 + (*p++ - '0');
    return x;
}
```

⟨*function* `ftprestart` *(hget.c)* 260c⟩≡ (261b)

```
int
ftprestart(int ctl, Out *out, URL *u, Range *r, long mtime)
{
    Tm tm;
    char msg[1024];
    long x, rmtime;

    ftpcmd(ctl, "MDTM %s", u->page);
    if(ftprcode(ctl, msg, sizeof(msg)) != Success){
        r->start = 0;
        return 0;       /* need to do something */
    }

    /* decode modification time */
    if(strlen(msg) < 4 + 4 + 2 + 2 + 2 + 2 + 2){
        r->start = 0;
        return 0;       /* need to do something */
    }
    memset(&tm, 0, sizeof(tm));
    tm.year = getdec(msg+4, 4) - 1900;
    tm.mon = getdec(msg+4+4, 2) - 1;
    tm.mday = getdec(msg+4+4+2, 2);
    tm.hour = getdec(msg+4+4+2+2, 2);
    tm.min = getdec(msg+4+4+2+2+2, 2);
    tm.sec = getdec(msg+4+4+2+2+2+2, 2);
    strcpy(tm.zone, "GMT");
    rmtime = tm2sec(&tm);
    if(rmtime > mtime)
        r->start = 0;
```

```
    /* get size */
    ftpcmd(ctl, "SIZE %s", u->page);
    if(ftprcode(ctl, msg, sizeof(msg)) == Success){
        x = atol(msg+4);
        if(r->start == x)
            return 1;   /* we're up to date */
        r->end = x;
    }

    /* seek to restart point */
    if(r->start > 0){
        ftpcmd(ctl, "REST %lud", r->start);
        if(ftprcode(ctl, msg, sizeof(msg)) == Incomplete){
            setoffset(out, r->start);
        }else
            r->start = 0;
    }

    return 0;   /* need to do something */
}
```

⟨*function* logon*(hget.c)* 261a⟩≡ (261b)

```
int
logon(int ctl)
{
    char msg[1024];

    /* login anonymous */
    ftpcmd(ctl, "USER anonymous");
    switch(ftprcode(ctl, msg, sizeof(msg))){
    case Success:
        return 0;
    case Incomplete:
        break;  /* need password */
    default:
        werrstr("USER: %s", msg);
        return Server;
    }

    /* send user id as password */
    sprint(msg, "%s@closedmind.org", getuser());
    ftpcmd(ctl, "PASS %s", msg);
    if(ftprcode(ctl, msg, sizeof(msg)) != Success){
        werrstr("PASS: %s", msg);
        return Server;
    }

    return 0;
}
```

⟨misc/hget.c 261b⟩≡

```
#include <u.h>
#include <libc.h>
#include <ctype.h>
#include <bio.h>
#include <ip.h>
#include <libsec.h>
#include <auth.h>
```

```
typedef struct URL URL;
```
⟨*struct* URL*(hget.c)* 247b⟩

```
typedef struct Range Range;
```
⟨*struct* Range*(hget.c)* 247c⟩

```
typedef struct Out Out;
```
⟨*struct* Out*(hget.c)* 248a⟩

```
enum
{
    Other,
    Http,
    Https,
    Ftp,
};

enum
{
    Eof = 0,
    Error = -1,
    Server = -2,
    Changed = -3,
};

bool debug;
char *ofile;


int doftp(URL*, URL*, Range*, Out*, long);
int dohttp(URL*, URL*,  Range*, Out*, long);
int crackurl(URL*, char*);
Range*  crackrange(char*);
int getheader(int, char*, int);
int httpheaders(int, int, URL*, Range*);
int httprcode(int);
int cistrncmp(char*, char*, int);
int cistrcmp(char*, char*);
void    initibuf(void);
int readline(int, char*, int);
int readibuf(int, char*, int);
int dfprint(int, char*, ...);
void    unreadline(char*);
int output(Out*, char*, int);
void    setoffset(Out*, int);

int verbose;
char    *net;
char    tcpdir[NETPATHLEN];
int headerprint;
```

⟨*global* method*(hget.c)* 248b⟩

⟨*function* usage*(hget.c)* 248c⟩

⟨*function* main*(hget.c)* 248d⟩

⟨*function* crackurl*(hget.c)* 250⟩

```
char *day[] = {
```

```
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};

char *month[] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};

struct
{
    int fd;
    long    mtime;
} note;

void
catch(void*, char*)
{
    Dir d;

    nulldir(&d);
    d.mtime = note.mtime;
    if(dirfwstat(note.fd, &d) < 0)
        sysfatal("catch: can't dirfwstat: %r");
    noted(NDFLT);
}
```

⟨*function* `dohttp` *(hget.c)* 251⟩

⟨*function* `httprcode` *(hget.c)* 255⟩

```
/* read in and crack the http headers, update u and r */
void    hhetag(char*, URL*, Range*);
void    hhmtime(char*, URL*, Range*);
void    hhclen(char*, URL*, Range*);
void    hhcrange(char*, URL*, Range*);
void    hhuri(char*, URL*, Range*);
void    hhlocation(char*, URL*, Range*);
void    hhauth(char*, URL*, Range*);
```

⟨*global* `headers` *(hget.c)* 256a⟩

⟨*function* `httheaders` *(hget.c)* 256b⟩

⟨*function* `getheader` *(hget.c)* 256c⟩

```
void
hhetag(char *p, URL *u, Range*)
{
    if(u->etag != nil){
        if(strcmp(u->etag, p) != 0)
            sysfatal("file changed underfoot");
    } else
        u->etag = strdup(p);
}

char*   monthchars = "janfebmaraprmayjunjulaugsepoctnovdec";

void
hhmtime(char *p, URL *u, Range*)
{
```

```
char *month, *day, *yr, *hms;
char *fields[6];
Tm tm, now;
int i;

i = getfields(p, fields, 6, 1, " \t");
if(i < 5)
    return;

day = fields[1];
month = fields[2];
yr = fields[3];
hms = fields[4];

/* default time */
now = *gmtime(time(0));
tm = now;
tm.yday = 0;

/* convert ascii month to a number twixt 1 and 12 */
if(*month >= '0' && *month <= '9'){
    tm.mon = atoi(month) - 1;
    if(tm.mon < 0 || tm.mon > 11)
        tm.mon = 5;
} else {
    for(p = month; *p; p++)
        *p = tolower(*p);
    for(i = 0; i < 12; i++)
        if(strncmp(&monthchars[i*3], month, 3) == 0){
            tm.mon = i;
            break;
        }
}

tm.mday = atoi(day);

if(hms) {
    tm.hour = strtoul(hms, &p, 10);
    if(*p == ':') {
        p++;
        tm.min = strtoul(p, &p, 10);
        if(*p == ':') {
            p++;
            tm.sec = strtoul(p, &p, 10);
        }
    }
    if(tolower(*p) == 'p')
        tm.hour += 12;
}

if(yr) {
    tm.year = atoi(yr);
    if(tm.year >= 1900)
        tm.year -= 1900;
} else {
    if(tm.mon > now.mon || (tm.mon == now.mon && tm.mday > now.mday+1))
        tm.year--;
}

strcpy(tm.zone, "GMT");
```

```
    /* convert to epoch seconds */
    u->mtime = tm2sec(&tm);
}

void
hhclen(char *p, URL*, Range *r)
{
    r->end = atoi(p);
}

void
hhcrange(char *p, URL*, Range *r)
{
    char *x;
    vlong l;

    l = 0;
    x = strchr(p, '/');
    if(x)
        l = atoll(x+1);
    if(l == 0) {
        x = strchr(p, '-');
        if(x)
            l = atoll(x+1);
    }
    if(l)
        r->end = l;
}

void
hhuri(char *p, URL *u, Range*)
{
    if(*p != '<')
        return;
    u->redirect = strdup(p+1);
    p = strchr(u->redirect, '>');
    if(p != nil)
        *p = 0;
}

void
hhlocation(char *p, URL *u, Range*)
{
    u->redirect = strdup(p);
}

void
hhauth(char *p, URL *u, Range*)
{
    char *f[4];
    UserPasswd *up;
    char *s, cred[64];

    if (cistrncmp(p, "basic ", 6) != 0)
        sysfatal("only Basic authentication supported");

    if (gettokens(p, f, nelem(f), "\"") < 2)
        sysfatal("garbled auth data");

    if ((up = auth_getuserpasswd(auth_getkey, "proto=pass service=http server=%q realm=%q",
```

```
            u->host, f[1])) == nil)
            sysfatal("cannot authenticate");

    s = smprint("%s:%s", up->user, up->passwd);
    if(enc64(cred, sizeof(cred), (uchar *)s, strlen(s)) == -1)
        sysfatal("enc64");
        free(s);

    assert(u->cred = strdup(cred));
}

enum
{
    /* ftp return codes */
    Extra=      1,
    Success=    2,
    Incomplete= 3,
    TempFail=   4,
    PermFail=   5,

    Nnetdir=    64, /* max length of network directory paths */
    Ndialstr=   64,     /* max length of dial strings */
};

int ftpcmd(int, char*, ...);
int ftprcode(int, char*, int);
int hello(int);
int logon(int);
int xfertype(int, char*);
int passive(int, URL*);
int active(int, URL*);
int ftpxfer(int, Out*, Range*);
int terminateftp(int, int);
int getaddrport(char*, uchar*, uchar*);
int ftprestart(int, Out*, URL*, Range*, long);
```

⟨*function* `doftp` *(hget.c)* 257⟩
⟨*function* `ftpcmt` *(hget.c)* 259a⟩
⟨*function* `ftprcode` *(hget.c)* 259b⟩

⟨*function* `hello` *(hget.c)* 260a⟩
⟨*function* `getdec` *(hget.c)* 260b⟩
⟨*function* `ftprestart` *(hget.c)* 260c⟩
⟨*function* `logon` *(hget.c)* 261a⟩

```
int
xfertype(int ctl, char *t)
{
    char msg[1024];

    ftpcmd(ctl, "TYPE %s", t);
    if(ftprcode(ctl, msg, sizeof(msg)) != Success){
        werrstr("TYPE %s: %s", t, msg);
        return Server;
    }

    return 0;
```

```
}

int
passive(int ctl, URL *u)
{
    char msg[1024];
    char ipaddr[32];
    char *f[6];
    char *p;
    int fd;
    int port;
    char aport[12];

    ftpcmd(ctl, "PASV");
    if(ftprcode(ctl, msg, sizeof(msg)) != Success)
        return Error;

    /* get address and port number from reply, this is AI */
    p = strchr(msg, '(');
    if(p == nil){
        for(p = msg+3; *p; p++)
            if(isdigit(*p))
                break;
    } else
        p++;
    if(getfields(p, f, 6, 0, ",)") < 6){
        werrstr("ftp protocol botch");
        return Server;
    }
    snprint(ipaddr, sizeof(ipaddr), "%s.%s.%s.%s",
        f[0], f[1], f[2], f[3]);
    port = ((atoi(f[4])&0xff)<<8) + (atoi(f[5])&0xff);
    sprint(aport, "%d", port);

    /* open data connection */
    fd = dial(netmkaddr(ipaddr, tcpdir, aport), 0, 0, 0);
    if(fd < 0){
        werrstr("passive mode failed: %r");
        return Error;
    }

    /* tell remote to send a file */
    ftpcmd(ctl, "RETR %s", u->page);
    if(ftprcode(ctl, msg, sizeof(msg)) != Extra){
        werrstr("RETR %s: %s", u->page, msg);
        return Error;
    }
    return fd;
}

int
active(int ctl, URL *u)
{
    char msg[1024];
    char dir[40], ldir[40];
    uchar ipaddr[4];
    uchar port[2];
    int lcfd, dfd, afd;

    /* announce a port for the call back */
```

```
    snprint(msg, sizeof(msg), "%s!*!0", tcpdir);
    afd = announce(msg, dir);
    if(afd < 0)
        return Error;

    /* get a local address/port of the annoucement */
    if(getaddrport(dir, ipaddr, port) < 0){
        close(afd);
        return Error;
    }

    /* tell remote side address and port*/
    ftpcmd(ctl, "PORT %d,%d,%d,%d,%d,%d", ipaddr[0], ipaddr[1], ipaddr[2],
        ipaddr[3], port[0], port[1]);
    if(ftprcode(ctl, msg, sizeof(msg)) != Success){
        close(afd);
        werrstr("active: %s", msg);
        return Error;
    }

    /* tell remote to send a file */
    ftpcmd(ctl, "RETR %s", u->page);
    if(ftprcode(ctl, msg, sizeof(msg)) != Extra){
        close(afd);
        werrstr("RETR: %s", msg);
        return Server;
    }

    /* wait for a connection */
    lcfd = listen(dir, ldir);
    if(lcfd < 0){
        close(afd);
        return Error;
    }
    dfd = accept(lcfd, ldir);
    if(dfd < 0){
        close(afd);
        close(lcfd);
        return Error;
    }
    close(afd);
    close(lcfd);

    return dfd;
}

int
ftpxfer(int in, Out *out, Range *r)
{
    char buf[1024];
    long vtime;
    int i, n;

    vtime = 0;
    for(n = 0;;n += i){
        i = read(in, buf, sizeof(buf));
        if(i == 0)
            break;
        if(i < 0)
            return Error;
```

```
        if(output(out, buf, i) != i)
            return Error;
        r->start += i;
        if(verbose && (vtime != time(0) || r->start == r->end)) {
            vtime = time(0);
            fprint(2, "%ld %ld\n", r->start, r->end);
        }
    }
    return n;
}

int
terminateftp(int ctl, int rv)
{
    close(ctl);
    return rv;
}

/*
 * case insensitive strcmp (why aren't these in libc?)
 */
int
cistrncmp(char *a, char *b, int n)
{
    while(n-- > 0){
        if(tolower(*a++) != tolower(*b++))
            return -1;
    }
    return 0;
}

int
cistrcmp(char *a, char *b)
{
    while(*a || *b)
        if(tolower(*a++) != tolower(*b++))
            return -1;

    return 0;
}

/*
 *  buffered io
 */
struct
{
    char *rp;
    char *wp;
    char buf[4*1024];
} b;

void
initibuf(void)
{
    b.rp = b.wp = b.buf;
}

/*
 *  read a possibly buffered line, strip off trailing while
 */
```

```
int
readline(int fd, char *buf, int len)
{
    int n;
    char *p;
    int eof = 0;

    len--;

    for(p = buf;;){
        if(b.rp >= b.wp){
            /* claude: b is styled like a ring (rp/wp pair + backing array),
             * but the original code never wraps -- wp only advances, and
             * read() always lands at wp without checking how much space is
             * left before b.buf+sizeof(b.buf). After enough bytes have
             * passed through, wp ends up within sizeof(b.buf)/2 of the end
             * and the next refill writes off the tail of the BSS region.
             * On small responses this never triggered, which is why the
             * bug also exists unfixed in plan9-0intro and plan9port.
             * en.wikipedia.org ships enough header bytes (set-cookie +
             * report-to JSON + nel JSON + strict-transport-security) to
             * push wp past the end.
             *
             * Reset both pointers to b.buf whenever the buffer drains --
             * rp catches wp only when there's no unread data, so we can
             * safely re-anchor. unreadline() independently re-anchors
             * rp/wp at b.buf when it fires, so the two mutators stay
             * consistent. */
            b.rp = b.wp = b.buf;
            n = read(fd, b.wp, sizeof(b.buf)/2);
            if(n < 0)
                return -1;
            if(n == 0){
                eof = 1;
                break;
            }
            b.wp += n;
        }
        n = *b.rp++;
        if(len > 0){
            *p++ = n;
            len--;
        }
        if(n == '\n')
            break;
    }

    /* drop trailing white */
    for(;;){
        if(p <= buf)
            break;
        n = *(p-1);
        if(n != ' ' && n != '\t' && n != '\r' && n != '\n')
            break;
        p--;
    }
    *p = 0;

    if(eof && p == buf)
        return -1;
```

```
    return p-buf;
}

void
unreadline(char *line)
{
    int i, n;

    i = strlen(line);
    n = b.wp-b.rp;
    memmove(&b.buf[i+1], b.rp, n);
    memmove(b.buf, line, i);
    b.buf[i] = '\n';
    b.rp = b.buf;
    b.wp = b.rp + i + 1 + n;
}

int
readibuf(int fd, char *buf, int len)
{
    int n;

    n = b.wp-b.rp;
    if(n > 0){
        if(n > len)
            n = len;
        memmove(buf, b.rp, n);
        b.rp += n;
        return n;
    }
    return read(fd, buf, len);
}

int
dfprint(int fd, char *fmt, ...)
{
    char buf[4*1024];
    va_list arg;

    va_start(arg, fmt);
    vseprint(buf, buf+sizeof(buf), fmt, arg);
    va_end(arg);
    if(debug)
        fprint(STDERR, "%d -> %s", fd, buf);
    return fprint(fd, "%s", buf);
}

int
getaddrport(char *dir, uchar *ipaddr, uchar *port)
{
    char buf[256];
    int fd, i;
    char *p;

    snprint(buf, sizeof(buf), "%s/local", dir);
    fd = open(buf, OREAD);
    if(fd < 0)
        return -1;
    i = read(fd, buf, sizeof(buf)-1);
```

```
    close(fd);
    if(i <= 0)
        return -1;
    buf[i] = 0;
    p = strchr(buf, '!');
    if(p != nil)
        *p++ = 0;
    v4parseip(ipaddr, buf);
    i = atoi(p);
    port[0] = i>>8;
    port[1] = i;
    return 0;
}

void
md5free(DigestState *state)
{
    uchar x[MD5dlen];
    md5(nil, 0, x, state);
}

DigestState*
md5dup(DigestState *state)
{
    char *p;

    p = md5pickle(state);
    if(p == nil)
        sysfatal("md5pickle: %r");
    state = md5unpickle(p);
    if(state == nil)
        sysfatal("md5unpickle: %r");
    free(p);
    return state;
}

void
setoffset(Out *out, int offset)
{
    md5free(out->curr);
    if(offset == 0)
        out->curr = md5(nil, 0, nil, nil);
    else
        out->curr = nil;
    out->offset = offset;
    out->written = offset;
    if(ofile != nil)
        if(seek(out->fd, offset, 0) != offset)
            sysfatal("seek: %r");
}

/*
 * write some output, discarding it (but keeping track)
 * if we've already written it. if we've gone backwards,
 * verify that everything previously written matches
 * that which would have been written from the current
 * output.
 */
int
output(Out *out, char *buf, int nb)
```

```
{
    int n, d;
    uchar m0[MD5dlen], m1[MD5dlen];

    n = nb;
    d = out->written - out->offset;
    assert(d >= 0);
    if(d > 0){
        if(n < d){
            if(out->curr != nil)
                md5((uchar*)buf, n, nil, out->curr);
            out->offset += n;
            return n;
        }
        if(out->curr != nil){
            md5((uchar*)buf, d, m0, out->curr);
            out->curr = nil;
            md5(nil, 0, m1, md5dup(out->hiwat));
            if(memcmp(m0, m1, MD5dlen) != 0){
                fprint(2, "integrity check failure at offset %d\n", out->written);
                return -1;
            }
        }
        buf += d;
        n -= d;
        out->offset += d;
    }
    if(n > 0){
        out->hiwat = md5((uchar*)buf, n, nil, out->hiwat);
        n = write(out->fd, buf, n);
        if(n > 0){
            out->offset += n;
            out->written += n;
        }
    }
    return n + d;
}
```

E.4 tcs/

tcs/8859.h

⟨tcs/8859.h 273⟩≡

```
long tab8859_1[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xaa,0xab,0xac,0xad,0xae,0xaf,
0xb0,0xb1,0xb2,0xb3,0xb4,0xb5,0xb6,0xb7,0xb8,0xb9,0xba,0xbb,0xbc,0xbd,0xbe,0xbf,
0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd,0xce,0xcf,
```

```
0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd,0xde,0xdf,
0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0xea,0xeb,0xec,0xed,0xee,0xef,
0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd,0xfe,0xff,
};

long tab8859_2[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x0104,0x02d8,0x0141,0x00a4,0x013d,0x015a,0x00a7,
0x00a8,0x0160,0x015e,0x0164,0x0179,0x00ad,0x017d,0x017b,
0x00b0,0x0105,0x02db,0x0142,0x00b4,0x013e,0x015b,0x02c7,
0x00b8,0x0161,0x015f,0x0165,0x017a,0x02dd,0x017e,0x017c,
0x0154,0x00c1,0x00c2,0x0102,0x00c4,0x0139,0x0106,0x00c7,
0x010c,0x00c9,0x0118,0x00cb,0x011a,0x00cd,0x00ce,0x010e,
0x0110,0x0143,0x0147,0x00d3,0x00d4,0x0150,0x00d6,0x00d7,
0x0158,0x016e,0x00da,0x0170,0x00dc,0x00dd,0x0162,0x00df,
0x0155,0x00e1,0x00e2,0x0103,0x00e4,0x013a,0x0107,0x00e7,
0x010d,0x00e9,0x0119,0x00eb,0x011b,0x00ed,0x00ee,0x010f,
0x0111,0x0144,0x0148,0x00f3,0x00f4,0x0151,0x00f6,0x00f7,
0x0159,0x016f,0x00fa,0x0171,0x00fc,0x00fd,0x0163,0x02d9,
};

long tab8859_3[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x0126,0x02d8,0x00a3,0x00a4,    -1,0x0124,0x00a7,
0x00a8,0x0130,0x015e,0x011e,0x0134,0x00ad,    -1,0x017b,
0x00b0,0x0127,0x00b2,0x00b3,0x00b4,0x00b5,0x0125,0x00b7,
0x00b8,0x0131,0x015f,0x011f,0x0135,0x00bd,    -1,0x017c,
0x00c0,0x00c1,0x00c2,    -1,0x00c4,0x010a,0x0108,0x00c7,
0x00c8,0x00c9,0x00ca,0x00cb,0x00cc,0x00cd,0x00ce,0x00cf,
    -1,0x00d1,0x00d2,0x00d3,0x00d4,0x0120,0x00d6,0x00d7,
0x011c,0x00d9,0x00da,0x00db,0x00dc,0x016c,0x015c,0x00df,
0x00e0,0x00e1,0x00e2,    -1,0x00e4,0x010b,0x0109,0x00e7,
0x00e8,0x00e9,0x00ea,0x00eb,0x00ec,0x00ed,0x00ee,0x00ef,
    -1,0x00f1,0x00f2,0x00f3,0x00f4,0x0121,0x00f6,0x00f7,
0x011d,0x00f9,0x00fa,0x00fb,0x00fc,0x016d,0x015d,0x02d9,
};

long tab8859_4[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
```

```
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x0104,0x0138,0x0156,0x00a4,0x0128,0x013b,0x00a7,
0x00a8,0x0160,0x0112,0x0122,0x0166,0x00ad,0x017d,0x00af,
0x00b0,0x0105,0x02db,0x0157,0x00b4,0x0129,0x013c,0x02c7,
0x00b8,0x0161,0x0113,0x0123,0x0167,0x014a,0x017e,0x014b,
0x0100,0x00c1,0x00c2,0x00c3,0x00c4,0x00c5,0x00c6,0x012e,
0x010c,0x00c9,0x0118,0x00cb,0x0116,0x00cd,0x00ce,0x012a,
0x0110,0x0145,0x014c,0x0136,0x00d4,0x00d5,0x00d6,0x00d7,
0x00d8,0x0172,0x00da,0x00db,0x00dc,0x0168,0x016a,0x00df,
0x0101,0x00e1,0x00e2,0x00e3,0x00e4,0x00e5,0x00e6,0x012f,
0x010d,0x00e9,0x0119,0x00eb,0x0117,0x00ed,0x00ee,0x012b,
0x0111,0x0146,0x014d,0x0137,0x00f4,0x00f5,0x00f6,0x00f7,
0x00f8,0x0173,0x00fa,0x00fb,0x00fc,0x0169,0x016b,0x02d9,
};

long tab8859_5[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x0401,0x0402,0x0403,0x0404,0x0405,0x0406,0x0407,
0x0408,0x0409,0x040a,0x040b,0x040c,0x00ad,0x040e,0x040f,
0x0410,0x0411,0x0412,0x0413,0x0414,0x0415,0x0416,0x0417,
0x0418,0x0419,0x041a,0x041b,0x041c,0x041d,0x041e,0x041f,
0x0420,0x0421,0x0422,0x0423,0x0424,0x0425,0x0426,0x0427,
0x0428,0x0429,0x042a,0x042b,0x042c,0x042d,0x042e,0x042f,
0x0430,0x0431,0x0432,0x0433,0x0434,0x0435,0x0436,0x0437,
0x0438,0x0439,0x043a,0x043b,0x043c,0x043d,0x043e,0x043f,
0x0440,0x0441,0x0442,0x0443,0x0444,0x0445,0x0446,0x0447,
0x0448,0x0449,0x044a,0x044b,0x044c,0x044d,0x044e,0x044f,
0x2116,0x0451,0x0452,0x0453,0x0454,0x0455,0x0456,0x0457,
0x0458,0x0459,0x045a,0x045b,0x045c,0x00a7,0x045e,0x045f,
};

long tab8859_6[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
```

```
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,    -1,    -1,    -1,0x00a4,    -1,    -1,    -1,
    -1,    -1,    -1,    -1,0x060c,0x00ad,    -1,    -1,
    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
    -1,    -1,    -1,0x061b,    -1,    -1,    -1,0x061f,
    -1,0x0621,0x0622,0x0623,0x0624,0x0625,0x0626,0x0627,
0x0628,0x0629,0x062a,0x062b,0x062c,0x062d,0x062e,0x062f,
0x0630,0x0631,0x0632,0x0633,0x0634,0x0635,0x0636,0x0637,
0x0638,0x0639,0x063a,    -1,    -1,    -1,    -1,    -1,
0x0640,0x0641,0x0642,0x0643,0x0644,0x0645,0x0646,0x0647,
0x0648,0x0649,0x064a,0x064b,0x064c,0x064d,0x064e,0x064f,
0x0650,0x0651,0x0652,    -1,    -1,    -1,    -1,    -1,
    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
};

long tab8859_7[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x2018,0x2019,0x00a3,    -1,    -1,0x00a6,0x00a7,
0x00a8,0x00a9,    -1,0x00ab,0x00ac,0x00ad,    -1,0x2015,
0x00b0,0x00b1,0x00b2,0x00b3,0x0384,0x0385,0x0386,0x00b7,
0x0388,0x0389,0x038a,0x00bb,0x038c,0x00bd,0x038e,0x038f,
0x0390,0x0391,0x0392,0x0393,0x0394,0x0395,0x0396,0x0397,
0x0398,0x0399,0x039a,0x039b,0x039c,0x039d,0x039e,0x039f,
0x03a0,0x03a1,    -1,0x03a3,0x03a4,0x03a5,0x03a6,0x03a7,
0x03a8,0x03a9,0x03aa,0x03ab,0x03ac,0x03ad,0x03ae,0x03af,
0x03b0,0x03b1,0x03b2,0x03b3,0x03b4,0x03b5,0x03b6,0x03b7,
0x03b8,0x03b9,0x03ba,0x03bb,0x03bc,0x03bd,0x03be,0x03bf,
0x03c0,0x03c1,0x03c2,0x03c3,0x03c4,0x03c5,0x03c6,0x03c7,
0x03c8,0x03c9,0x03ca,0x03cb,0x03cc,0x03cd,0x03ce,    -1,
};

long tab8859_8[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,    -1,0x00a2,0x00a3,0x00a4,0x00a5,0x00a6,0x00a7,
0x00a8,0x00a9,0x00d7,0x00ab,0x00ac,0x00ad,0x00ae,0x203e,
0x00b0,0x00b1,0x00b2,0x00b3,0x00b4,0x00b5,0x00b6,0x00b7,
0x00b8,0x00b9,0x00f7,0x00bb,0x00bc,0x00bd,0x00be,    -1,
    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
```

```
    -1,    -1,    -1,    -1,    -1,    -1,    -1,0x2017,
0x05d0,0x05d1,0x05d2,0x05d3,0x05d4,0x05d5,0x05d6,0x05d7,
0x05d8,0x05d9,0x05da,0x05db,0x05dc,0x05dd,0x05de,0x05df,
0x05e0,0x05e1,0x05e2,0x05e3,0x05e4,0x05e5,0x05e6,0x05e7,
0x05e8,0x05e9,0x05ea,    -1,    -1,    -1,    -1,    -1,
};

long tab8859_9[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x00a1,0x00a2,0x00a3,0x00a4,0x00a5,0x00a6,0x00a7,
0x00a8,0x00a9,0x00aa,0x00ab,0x00ac,0x00ad,0x00ae,0x00af,
0x00b0,0x00b1,0x00b2,0x00b3,0x00b4,0x00b5,0x00b6,0x00b7,
0x00b8,0x00b9,0x00ba,0x00bb,0x00bc,0x00bd,0x00be,0x00bf,
0x00c0,0x00c1,0x00c2,0x00c3,0x00c4,0x00c5,0x00c6,0x00c7,
0x00c8,0x00c9,0x00ca,0x00cb,0x00cc,0x00cd,0x00ce,0x00cf,
0x011e,0x00d1,0x00d2,0x00d3,0x00d4,0x00d5,0x00d6,0x00d7,
0x00d8,0x00d9,0x00da,0x00db,0x00dc,0x0130,0x015e,0x00df,
0x00e0,0x00e1,0x00e2,0x00e3,0x00e4,0x00e5,0x00e6,0x00e7,
0x00e8,0x00e9,0x00ea,0x00eb,0x00ec,0x00ed,0x00ee,0x00ef,
0x011f,0x00f1,0x00f2,0x00f3,0x00f4,0x00f5,0x00f6,0x00f7,
0x00f8,0x00f9,0x00fa,0x00fb,0x00fc,0x0131,0x015f,0x00ff,
};

long tab8859_10[256] =  /* from dkuug.dk:i18n/charmaps/ISO_8859-10:1993 */
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0x00a0,0x0104,0x0112,0x0122,0x012a,0x0128,0x0136,0x00a7,
0x013b,0x0110,0x0160,0x0166,0x017d,0x00ad,0x016a,0x014a,
0x00b0,0x0105,0x0113,0x0123,0x012b,0x0129,0x0137,0x00b7,
0x013c,0x0110,0x0161,0x0167,0x017e,0x2014,0x016b,0x014b,
0x0100,0x00c1,0x00c2,0x00c3,0x00c4,0x00c5,0x00c6,0x012e,
0x010c,0x00c9,0x0118,0x00cb,0x0116,0x00cd,0x00ce,0x00cf,
0x00d0,0x0145,0x014c,0x00d3,0x00d4,0x00d5,0x00d6,0x0168,
0x00d8,0x0172,0x00da,0x00db,0x00dc,0x00dd,0x00de,0x00df,
0x0101,0x00e1,0x00e2,0x00e3,0x00e4,0x00e5,0x00e6,0x012f,
0x010d,0x00e9,0x0119,0x00eb,0x0117,0x00ed,0x00ee,0x00ef,
0x00f0,0x0146,0x014d,0x00f3,0x00f4,0x00f5,0x00f6,0x0169,
0x00f8,0x0173,0x00fa,0x00fb,0x00fc,0x00fd,0x00fe,0x0138,
};
```

```
long tab8859_15[256] =      /* from anyrhine@cs.helsinki.fi (Aki Nyrhinen) */
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
0xa0,0xa1,0xa2,0xa3,0x20ac,0xa5,0x0160,0xa7,0x0161,0xa9,0xaa,0xab,0xac,0xad,
0xae,0xaf,0xb0,0xb1,0xb2,0xb3,0x017d,0xb5,0xb6,0xb7,0x017e,0xb9,0xba,0xbb,
0x0152,0x0153,0x0178,0xbf,0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,
0xca,0xcb,0xcc,0xcd,0xce,0xcf,0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,
0xd9,0xda,0xdb,0xdc,0xdd,0xde,0xdf,0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,
0xe8,0xe9,0xea,0xeb,0xec,0xed,0xee,0xef,0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,
0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd,0xfe,0xff
};
```

tcs/conv.h

⟨*macro* emit*(tcs)* 278a⟩≡ (278c)

```
#define     emit(x)     *(*r)++ = (x)
```

⟨*constant* NRUNE*(tcs)* 278b⟩≡ (278c)

```
#define     NRUNE       (Runemax+1)
```

⟨tcs/conv.h 278c⟩≡

```
void jis_in(int fd, long *notused, struct convert *out);
void jisjis_in(int fd, long *notused, struct convert *out);
void msjis_in(int fd, long *notused, struct convert *out);
void ujis_in(int fd, long *notused, struct convert *out);
void jisjis_out(Rune *base, int n, long *notused);
void ujis_out(Rune *base, int n, long *notused);
void msjis_out(Rune *base, int n, long *notused);
void big5_in(int fd, long *notused, struct convert *out);
void big5_out(Rune *base, int n, long *notused);
void gb_in(int fd, long *notused, struct convert *out);
void gb_out(Rune *base, int n, long *notused);
void gbk_in(int fd, long *notused, struct convert *out);
void gbk_out(Rune *base, int n, long *notused);
void uksc_in(int fd, long *notused, struct convert *out);
void uksc_out(Rune *base, int n, long *notused);
void html_in(int fd, long *notused, struct convert *out);
void html_out(Rune *base, int n, long *notused);
void tune_in(int fd, long *notused, struct convert *out);
void tune_out(Rune *base, int n, long *notused);

⟨macro emit(tcs) 278a⟩
⟨constant NRUNE(tcs) 278b⟩

extern long tab[];      /* common table indexed by Runes for reverse mappings */
```

tcs/hdr.h

⟨*constant* N*(tcs)* 278d⟩≡ (279j)

```
#define     N       10000       /* just blocking */
```

⟨*macro* OUT*(tcs)* 279a⟩≡ (279j)
```
#define OUT(out, r, n)  if(out->flags&Table) outtable(r, n, (long *)out->data);\
            else ((Outfn)(out->fn))(r, n, (long *)0)
```

⟨*constant* BADMAP*(tcs)* 279b⟩≡ (279j)
```
#define     BADMAP      (0xFFFD)
```

⟨*constant* BYTEBADMAP*(tcs)* 279c⟩≡ (279j)
```
#define     BYTEBADMAP  ('?')       /* badmap but has to fit in a byte */
```

⟨*constant* ESC*(tcs)* 279d⟩≡ (279j)
```
#define     ESC     033
```

⟨*constant* EPR*(tcs)* 279e⟩≡ (279j)
```
#define EPR     fprint(2,
```

⟨*macro* EXIT*(tcs)* 279f⟩≡ (279j)
```
#define EXIT(n,s)   exits(s)
```

⟨*constant* EPR (tcs/hdr.h)*(tcs)* 279g⟩≡ (279j)
```
#define EPR     fprintf(stderr,
```

⟨*macro* USED*(tcs)* 279h⟩≡ (279j)
```
#define USED(x)     /* in plan 9, USED(x) tells the compiler to treat x as used */
```

⟨*macro* EXIT (tcs/hdr.h)*(tcs)* 279i⟩≡ (279j)
```
#define EXIT(n,s)   exit(n)
```

⟨tcs/hdr.h 279j⟩≡
```
extern int squawk;
extern int clean;
extern char *file;
extern int verbose;
extern long ninput, noutput, nrunes, nerrors;

enum { From = 1, Table = 2, Func = 4 };

typedef void (*Fnptr)(void);
struct convert{
    char *name;
    char *chatter;
    int flags;
    void *data;
    Fnptr fn;
};
extern struct convert convert[];
struct convert *conv(char *, int);
typedef void (*Infn)(int, long *, struct convert *);
typedef void (*Outfn)(Rune *, int, long *);
void outtable(Rune *, int, long *);

void utf_in(int, long *, struct convert *);
void utf_out(Rune *, int, long *);
void isoutf_in(int, long *, struct convert *);
void isoutf_out(Rune *, int, long *);

⟨constant N(tcs) 278d⟩
⟨macro OUT(tcs) 279a⟩

extern Rune runes[N];
```

```
extern char obuf[UTFmax*N]; /* maximum bloat from N runes */
```

⟨*constant* BADMAP*(tcs)* 279b⟩
⟨*constant* BYTEBADMAP*(tcs)* 279c⟩
⟨*constant* ESC*(tcs)* 279d⟩

```
#ifdef  PLAN9
```
⟨*constant* EPR*(tcs)* 279e⟩
⟨*macro* EXIT*(tcs)* 279f⟩
```
#else
```
⟨*constant* EPR (tcs/hdr.h)*(tcs)* 279g⟩
⟨*macro* USED*(tcs)* 279h⟩
⟨*macro* EXIT (tcs/hdr.h)*(tcs)* 279i⟩
```
#endif
```

tcs/html.c

⟨*struct* Hchar*(tcs)* 280a⟩≡ (288a)
```
struct Hchar
{
    char *s;
    Rune r;
};
```

⟨*global* byname*(tcs)* 280b⟩≡ (288a)
```
/*
 * Names beginning with _ are names we recognize
 * (without the underscore) but will not generate,
 * because they are nonstandard.
 */
static Hchar byname[] =
{
    {"AElig", 198},
    {"Aacute", 193},
    {"Acirc", 194},
    {"Agrave", 192},
    {"Alpha", 913},
    {"Aring", 197},
    {"Atilde", 195},
    {"Auml", 196},
    {"Beta", 914},
    {"Ccedil", 199},
    {"Chi", 935},
    {"Dagger", 8225},
    {"Delta", 916},
    {"ETH", 208},
    {"Eacute", 201},
    {"Ecirc", 202},
    {"Egrave", 200},
    {"Epsilon", 917},
    {"Eta", 919},
    {"Euml", 203},
    {"Gamma", 915},
    {"Iacute", 205},
    {"Icirc", 206},
    {"Igrave", 204},
    {"Iota", 921},
    {"Iuml", 207},
    {"Kappa", 922},
```

```
{"Lambda", 923},
{"Mu", 924},
{"Ntilde", 209},
{"Nu", 925},
{"OElig", 338},
{"Oacute", 211},
{"Ocirc", 212},
{"Ograve", 210},
{"Omega", 937},
{"Omicron", 927},
{"Oslash", 216},
{"Otilde", 213},
{"Ouml", 214},
{"Phi", 934},
{"Pi", 928},
{"Prime", 8243},
{"Psi", 936},
{"Rho", 929},
{"Scaron", 352},
{"Sigma", 931},
{"THORN", 222},
{"Tau", 932},
{"Theta", 920},
{"Uacute", 218},
{"Ucirc", 219},
{"Ugrave", 217},
{"Upsilon", 933},
{"Uuml", 220},
{"Xi", 926},
{"Yacute", 221},
{"Yuml", 376},
{"Zeta", 918},
{"aacute", 225},
{"acirc", 226},
{"acute", 180},
{"aelig", 230},
{"agrave", 224},
{"alefsym", 8501},
{"alpha", 945},
{"amp", 38},
{"and", 8743},
{"ang", 8736},
{"aring", 229},
{"asymp", 8776},
{"atilde", 227},
{"auml", 228},
{"bdquo", 8222},
{"beta", 946},
{"brvbar", 166},
{"bull", 8226},
{"cap", 8745},
{"ccedil", 231},
{"cdots", 8943},
{"cedil", 184},
{"cent", 162},
{"chi", 967},
{"circ", 710},
{"clubs", 9827},
{"cong", 8773},
{"copy", 169},
```

```
{"crarr", 8629},
{"cup", 8746},
{"curren", 164},
{"dArr", 8659},
{"dagger", 8224},
{"darr", 8595},
{"ddots", 8945},
{"deg", 176},
{"delta", 948},
{"diams", 9830},
{"divide", 247},
{"eacute", 233},
{"ecirc", 234},
{"egrave", 232},
{"_emdash", 8212},  /* non-standard but commonly used */
{"empty", 8709},
{"emsp", 8195},
{"_endash", 8211},  /* non-standard but commonly used */
{"ensp", 8194},
{"epsilon", 949},
{"equiv", 8801},
{"eta", 951},
{"eth", 240},
{"euml", 235},
{"euro", 8364},
{"exist", 8707},
{"fnof", 402},
{"forall", 8704},
{"frac12", 189},
{"frac14", 188},
{"frac34", 190},
{"frasl", 8260},
{"gamma", 947},
{"ge", 8805},
{"gt", 62},
{"hArr", 8660},
{"harr", 8596},
{"hearts", 9829},
{"hellip", 8230},
{"iacute", 237},
{"icirc", 238},
{"iexcl", 161},
{"igrave", 236},
{"image", 8465},
{"infin", 8734},
{"int", 8747},
{"iota", 953},
{"iquest", 191},
{"isin", 8712},
{"iuml", 239},
{"kappa", 954},
{"lArr", 8656},
{"lambda", 955},
{"lang", 9001},
{"laquo", 171},
{"larr", 8592},
{"lceil", 8968},
{"_ldots", 8230},
{"ldquo", 8220},
{"le", 8804},
```

```
{"lfloor", 8970},
{"lowast", 8727},
{"loz", 9674},
{"lrm", 8206},
{"lsaquo", 8249},
{"lsquo", 8216},
{"lt", 60},
{"macr", 175},
{"mdash", 8212},
{"micro", 181},
{"middot", 183},
{"minus", 8722},
{"mu", 956},
{"nabla", 8711},
{"nbsp", 160},
{"ndash", 8211},
{"ne", 8800},
{"ni", 8715},
{"not", 172},
{"notin", 8713},
{"nsub", 8836},
{"ntilde", 241},
{"nu", 957},
{"oacute", 243},
{"ocirc", 244},
{"oelig", 339},
{"ograve", 242},
{"oline", 8254},
{"omega", 969},
{"omicron", 959},
{"oplus", 8853},
{"or", 8744},
{"ordf", 170},
{"ordm", 186},
{"oslash", 248},
{"otilde", 245},
{"otimes", 8855},
{"ouml", 246},
{"para", 182},
{"part", 8706},
{"permil", 8240},
{"perp", 8869},
{"phi", 966},
{"pi", 960},
{"piv", 982},
{"plusmn", 177},
{"pound", 163},
{"prime", 8242},
{"prod", 8719},
{"prop", 8733},
{"psi", 968},
{"quad", 8193},
{"quot", 34},
{"rArr", 8658},
{"radic", 8730},
{"rang", 9002},
{"raquo", 187},
{"rarr", 8594},
{"rceil", 8969},
{"rdquo", 8221},
```

```
    {"real", 8476},
    {"reg", 174},
    {"rfloor", 8971},
    {"rho", 961},
    {"rlm", 8207},
    {"rsaquo", 8250},
    {"rsquo", 8217},
    {"sbquo", 8218},
    {"scaron", 353},
    {"sdot", 8901},
    {"sect", 167},
    {"shy", 173},
    {"sigma", 963},
    {"sigmaf", 962},
    {"sim", 8764},
    {"_sp", 8194},
    {"spades", 9824},
    {"sub", 8834},
    {"sube", 8838},
    {"sum", 8721},
    {"sup", 8835},
    {"sup1", 185},
    {"sup2", 178},
    {"sup3", 179},
    {"supe", 8839},
    {"szlig", 223},
    {"tau", 964},
    {"there4", 8756},
    {"theta", 952},
    {"thetasym", 977},
    {"thinsp", 8201},
    {"thorn", 254},
    {"tilde", 732},
    {"times", 215},
    {"trade", 8482},
    {"uArr", 8657},
    {"uacute", 250},
    {"uarr", 8593},
    {"ucirc", 251},
    {"ugrave", 249},
    {"uml", 168},
    {"upsih", 978},
    {"upsilon", 965},
    {"uuml", 252},
    {"_varepsilon", 8712},
    {"varphi", 981},
    {"_varpi", 982},
    {"varrho", 1009},
    {"vdots", 8942},
    {"_vsigma", 962},
    {"_vtheta", 977},
    {"weierp", 8472},
    {"xi", 958},
    {"yacute", 253},
    {"yen", 165},
    {"yuml", 255},
    {"zeta", 950},
    {"zwj", 8205},
    {"zwnj", 8204}
};
```

⟨*function* hnamecmp*(tcs)* 285a⟩≡ (288a)

```
static int
hnamecmp(const void *va, const void *vb)
{
    Hchar *a, *b;

    a = (Hchar*)va;
    b = (Hchar*)vb;
    return strcmp(a->s, b->s);
}
```

⟨*function* hrunecmp*(tcs)* 285b⟩≡ (288a)

```
static int
hrunecmp(const void *va, const void *vb)
{
    Hchar *a, *b;

    a = (Hchar*)va;
    b = (Hchar*)vb;
    return a->r - b->r;
}
```

⟨*function* html_init*(tcs)* 285c⟩≡ (288a)

```
static void
html_init(void)
{
    static int init;
    int i;

    if(init)
        return;
    init = 1;
    memmove(byrune, byname, sizeof byrune);

    /* Eliminate names we aren't allowed to generate. */
    for(i=0; i<nelem(byrune); i++){
        if(byrune[i].s[0] == '_'){
            byrune[i].r = Runeerror;
            byname[i].s++;
        }
    }

    qsort(byname, nelem(byname), sizeof byname[0], hnamecmp);
    qsort(byrune, nelem(byrune), sizeof byrune[0], hrunecmp);
}
```

⟨*function* findbyname*(tcs)* 285d⟩≡ (288a)

```
static Rune
findbyname(char *s)
{
    Hchar *h;
    int n, m, x;

    h = byname;
    n = nelem(byname);
    while(n > 0){
        m = n/2;
        x = strcmp(h[m].s, s);
        if(x == 0)
            return h[m].r;
```

```
        if(x < 0){
            h += m+1;
            n -= m+1;
        }else
            n = m;
    }
    return Runeerror;
}
```

⟨*function* `findbyrune`*(tcs)* 286a⟩≡ (288a)

```
static char*
findbyrune(Rune r)
{
    Hchar *h;
    int n, m;

    if(r == Runeerror)
        return nil;
    h = byrune;
    n = nelem(byrune);
    while(n > 0){
        m = n/2;
        if(h[m].r == r)
            return h[m].s;
        if(h[m].r < r){
            h += m+1;
            n -= m+1;
        }else
            n = m;
    }
    return nil;
}
```

⟨*function* `html_in`*(tcs)* 286b⟩≡ (288a)

```
void
html_in(int fd, long *x, struct convert *out)
{
    char buf[100], *p;
    Biobuf b;
    Rune rbuf[N];
    Rune *r, *er;
    int c, i;

    USED(x);

    html_init();
    r = rbuf;
    er = rbuf+N;
    Binit(&b, fd, OREAD);
    while((c = Bgetrune(&b)) != Beof){
        if(r >= er){
            OUT(out, rbuf, r-rbuf);
            r = rbuf;
        }
        if(c == '&'){
            buf[0] = c;
            for(i=1; i<nelem(buf)-1;){
                c = Bgetc(&b);
                if(c == Beof)
                    break;
```

```
                buf[i++] = c;
                if(strchr("; \t\r\n", c))
                    break;
            }
            buf[i] = 0;
            if(buf[i-1] == ';'){
                buf[i-1] = 0;
                if((c = findbyname(buf+1)) != Runeerror){
                    *r++ = c;
                    continue;
                }
                buf[i-1] = ';';
                if(buf[1] == '#'){
                    if(buf[2] == 'x')
                        c = strtol(buf+3, &p, 16);
                    else
                        c = strtol(buf+2, &p, 10);
                    if(*p != ';' || c >= NRUNE || c < 0)
                        goto bad;
                    *r++ = c;
                    continue;
                }
            }
        bad:
            for(p=buf; p<buf+i; ){
                p += chartorune(r++, p);
                if(r >= er){
                    OUT(out, rbuf, r-rbuf);
                    r = rbuf;
                }
            }
            continue;
        }
        *r++ = c;
    }
    if(r > rbuf)
        OUT(out, rbuf, r-rbuf);
    OUT(out, rbuf, 0);
}
```

⟨*function* `html_out` *(tcs)* 287⟩≡ (288a)

```
/*
 * use biobuf because can use more than UTFmax bytes per rune
 */
void
html_out(Rune *r, int n, long *x)
{
    char *s;
    Biobuf b;
    Rune *er;

    USED(x);
    html_init();
    Binit(&b, 1, OWRITE);
    er = r+n;
    for(; r<er; r++){
        if(*r < Runeself)
            Bputrune(&b, *r);
        else if((s = findbyrune(*r)) != nil)
            Bprint(&b, "&%s;", s);
```

```
        else
            Bprint(&b, "&#%d;", *r);
    }
    Bflush(&b);
}
```

⟨tcs/html.c 288a⟩≡
```
#include <u.h>
#include <libc.h>
#include <bio.h>
#include "hdr.h"
#include "conv.h"

typedef struct Hchar Hchar;
⟨struct Hchar(tcs) 280a⟩

/* &lt;, &gt;, &quot;, &amp; intentionally omitted */

⟨global byname(tcs) 280b⟩

static Hchar byrune[nelem(byname)];

⟨function hnamecmp(tcs) 285a⟩

⟨function hrunecmp(tcs) 285b⟩

⟨function html_init(tcs) 285c⟩

⟨function findbyname(tcs) 285d⟩

⟨function findbyrune(tcs) 286a⟩

⟨function html_in(tcs) 286b⟩

⟨function html_out(tcs) 287⟩
```

tcs/plan9.h

⟨constant Runeerror(tcs) 288b⟩≡ (289d)
```
#define     Runeerror   0x80    /* decoding error in UTF */
```

⟨constant Runeself(tcs) 288c⟩≡ (289d)
```
#define     Runeself    0x80    /* rune and UTF sequences are the same (<) */
```

⟨constant UTFmax(tcs) 288d⟩≡ (289d)
```
#define     UTFmax      6   /* maximum bytes per rune */
```

⟨constant ARGBEGIN(tcs) 288e⟩≡ (289d)
```
#define ARGBEGIN    for((argv0? 0: (argv0= *argv)),argv++,argc--;\
                argv[0] && argv[0][0]=='-' && argv[0][1];\
                argc--, argv++) {\
                char *_args, *_argt, _argc;\
                _args = &argv[0][1];\
                if(_args[0]=='-' && _args[1]==0){\
                    argc--; argv++; break;\
                }\
                _argc=0;while(*_args) switch(_argc= *_args++)
```

⟨*constant* ARGEND*(tcs)* 289a⟩≡ (289d)
```
#define ARGEND      }
```

⟨*macro* ARGF*(tcs)* 289b⟩≡ (289d)
```
#define ARGF()      (_argt=_args, _args="",\
                (*_argt? _argt: argv[1]? (argc--, *++argv): 0))
```

⟨*macro* ARGC*(tcs)* 289c⟩≡ (289d)
```
#define ARGC()      _argc
```

⟨tcs/plan9.h 289d⟩≡
```
typedef unsigned long Rune;     /* 21 bits */
typedef unsigned char uchar;
⟨constant Runeerror(tcs) 288b⟩
⟨constant Runeself(tcs) 288c⟩
⟨constant UTFmax(tcs) 288d⟩

/*
    plan 9 argument parsing
*/
⟨constant ARGBEGIN(tcs) 288e⟩
⟨constant ARGEND(tcs) 289a⟩
⟨macro ARGF(tcs) 289b⟩
⟨macro ARGC(tcs) 289c⟩
extern char *argv0;
```

tcs/utf.c

⟨*function* utf_in*(tcs)* 289e⟩≡ (296)
```
void
utf_in(int fd, long *notused, struct convert *out)
{
    char buf[N];
    int i, j, c, n, tot;
    ulong l;

    USED(notused);
    tot = 0;
    while((n = read(fd, buf+tot, N-tot)) >= 0){
        tot += n;
        for(i=j=0; i<=tot-UTFmax || (i<tot && (n==0 || fullrune(buf+i, tot-i))); ){
            c = our_mbtowc(&l, buf+i, tot-i);
            if(c == -1){
                if(squawk)
                    EPR "%s: bad UTF sequence near byte %ld in input\n", argv0, ninput+i);
                if(clean){
                    i++;
                    continue;
                }
                nerrors++;
                l = Runeerror;
                c = 1;
            }
            runes[j++] = l;
            i += c;
        }
        OUT(out, runes, j);
        tot -= i;
        ninput += i;
```

```
        if(tot)
            memmove(buf, buf+i, tot);
        if(n == 0)
            break;
    }
    OUT(out, runes, 0);
}
```

⟨*function* utf_out*(tcs)* 290a⟩≡ (296)

```
void
utf_out(Rune *base, int n, long *notused)
{
    char *p;
    Rune *r;

    USED(notused);
    nrunes += n;
    for(r = base, p = obuf; n-- > 0; r++){
        p += our_wctomb(p, *r);
    }
    noutput += p-obuf;
    write(1, obuf, p-obuf);
}
```

⟨*function* isoutf_in*(tcs)* 290b⟩≡ (296)

```
void
isoutf_in(int fd, long *notused, struct convert *out)
{
    char buf[N];
    int i, j, c, n, tot;

    USED(notused);
    tot = 0;
    while((n = read(fd, buf+tot, N-tot)) >= 0){
        tot += n;
        for(i=j=0; i<tot; ){
            if(!fullisorune(buf+i, tot-i))
                break;
            c = isochartorune(&runes[j], buf+i);
            if(runes[j] == Runeerror && c == 1){
                if(squawk)
                    EPR "%s: bad UTF sequence near byte %ld in input\n", argv0, ninput+i);
                if(clean){
                    i++;
                    continue;
                }
                nerrors++;
            }
            j++;
            i += c;
        }
        OUT(out, runes, j);
        tot -= i;
        ninput += i;
        if(tot)
            memmove(buf, buf+i, tot);
        if(n == 0)
            break;
    }
    OUT(out, runes, 0);
}
```

⟨*function* `isoutf_out`*(tcs)* 291a⟩≡ (296)

```
void
isoutf_out(Rune *base, int n, long *notused)
{
    char *p;
    Rune *r;

    USED(notused);
    nrunes += n;
    for(r = base, p = obuf; n-- > 0; r++)
        p += runetoisoutf(p, r);
    noutput += p-obuf;
    write(1, obuf, p-obuf);
}
```

⟨*function* `mktable`*(tcs)* 291b⟩≡ (296)

```
static
void
mktable(void)
{
    int i, u;

    for(i=0; i<256; i++) {
        u = i + (0x5E - 0xA0);
        if(i < 0xA0)
            u = i + (0xDF - 0x7F);
        if(i < 0x7F)
            u = i + (0x00 - 0x21);
        if(i < 0x21)
            u = i + (0xBE - 0x00);
        U[i] = u;
        T[u] = i;
    }
}
```

⟨*function* `isochartorune`*(tcs)* 291c⟩≡ (296)

```
int
isochartorune(Rune *rune, char *str)
{
    int c, c1, c2;
    long l;

    if(U[0] == 0)
        mktable();

    /*
     * one character sequence
     *  00000-0009F => 00-9F
     */
    c = *(uchar*)str;
    if(c < Char1) {
        *rune = c;
        return 1;
    }

    /*
     * two character sequence
     *  000A0-000FF => A0; A0-FF
     */
    c1 = *(uchar*)(str+1);
```

```
    if(c < Char21) {
        if(c1 >= Rune1 && c1 < Rune21) {
            *rune = c1;
            return 2;
        }
        goto bad;
    }

    /*
     * two character sequence
     *  00100-04015 => A1-F5; 21-7E/A0-FF
     */
    c1 = U[c1];
    if(c1 >= Esc)
        goto bad;
    if(c < Char22) {
        *rune =  (c-Char21)*Esc + c1 + Rune21;
        return 2;
    }

    /*
     * three character sequence
     *  04016-38E2D => A6-FB; 21-7E/A0-FF
     */
    c2 = U[*(uchar*)(str+2)];
    if(c2 >= Esc)
        goto bad;
    if(c < Char3) {
        l = (c-Char22)*Esc*Esc + c1*Esc + c2 + Rune22;
        if(l >= Rune3)
            goto bad;
        *rune = l;
        return 3;
    }

    /*
     * bad decoding
     */
bad:
    *rune = Bad;
    return 1;
}
```

⟨*function* `runetoisoutf` *(tcs)* 292⟩≡ (296)

```
int
runetoisoutf(char *str, Rune *rune)
{
    long c;

    if(T[0] == 0)
        mktable();

    /*
     * one character sequence
     *  00000-0009F => 00-9F
     */
    c = *rune;
    if(c < Rune1) {
        str[0] = c;
        return 1;
```

```
    }

    /*
     * two character sequence
     *  000A0-000FF => A0; A0-FF
     */
    if(c < Rune21) {
        str[0] = Char1;
        str[1] = c;
        return 2;
    }

    /*
     * two character sequence
     *  00100-04015 => A1-F5; 21-7E/A0-FF
     */
    if(c < Rune22) {
        c -= Rune21;
        str[0] = c/Esc + Char21;
        str[1] = T[c%Esc];
        return 2;
    }

    /*
     * three character sequence
     *  04016-38E2D => A6-FB; 21-7E/A0-FF
     */
    c -= Rune22;
    str[0] = c/(Esc*Esc) + Char22;
    str[1] = T[c/Esc%Esc];
    str[2] = T[c%Esc];
    return 3;
}
```

⟨*function* `fullisorune`*(tcs)* 293a⟩≡ (296)

```
int
fullisorune(char *str, int n)
{
    int c;

    if(n > 0) {
        c = *(uchar*)str;
        if(c < Char1)
            return 1;
        if(n > 1)
            if(c < Char22 || n > 2)
                return 1;
    }
    return 0;
}
```

⟨*function* `our_wctomb`*(tcs)* 293b⟩≡ (296)

```
int
our_wctomb(char *s, unsigned long wc)
{
    if(s == 0)
        return 0;       /* no shift states */
    if(wc & ~Wchar2) {
        if(wc & ~Wchar4) {
            if(wc & ~Wchar5) {
```

```
                /* 6 bytes */
                s[0] = T6 | ((wc >> 5*Bitx) & Mask6);
                s[1] = Tx | ((wc >> 4*Bitx) & Maskx);
                s[2] = Tx | ((wc >> 3*Bitx) & Maskx);
                s[3] = Tx | ((wc >> 2*Bitx) & Maskx);
                s[4] = Tx | ((wc >> 1*Bitx) & Maskx);
                s[5] = Tx |  (wc & Maskx);
                return 6;
            }
            /* 5 bytes */
            s[0] = T5 |  (wc >> 4*Bitx);
            s[1] = Tx | ((wc >> 3*Bitx) & Maskx);
            s[2] = Tx | ((wc >> 2*Bitx) & Maskx);
            s[3] = Tx | ((wc >> 1*Bitx) & Maskx);
            s[4] = Tx |  (wc & Maskx);
            return 5;
        }
        if(wc & ~Wchar3) {
            /* 4 bytes */
            s[0] = T4 |  (wc >> 3*Bitx);
            s[1] = Tx | ((wc >> 2*Bitx) & Maskx);
            s[2] = Tx | ((wc >> 1*Bitx) & Maskx);
            s[3] = Tx |  (wc & Maskx);
            return 4;
        }
        /* 3 bytes */
        s[0] = T3 |  (wc >> 2*Bitx);
        s[1] = Tx | ((wc >> 1*Bitx) & Maskx);
        s[2] = Tx |  (wc & Maskx);
        return 3;
    }
    if(wc & ~Wchar1) {
        /* 2 bytes */
        s[0] = T2 | (wc >> 1*Bitx);
        s[1] = Tx | (wc & Maskx);
        return 2;
    }
    /* 1 byte */
    s[0] = T1 | wc;
    return 1;
}
```

⟨*function* our_mbtowc*(tcs)* 294⟩≡ (296)

```
int
our_mbtowc(unsigned long *p, char *s, unsigned n)
{
    uchar *us;
    int c0, c1, c2, c3, c4, c5;
    unsigned long wc;

    if(s == 0)
        return 0;       /* no shift states */

    if(n < 1)
        goto bad;
    us = (uchar*)s;
    c0 = us[0];
    if(c0 >= T3) {
        if(n < 3)
            goto bad;
```

```
    c1 = us[1] ^ Tx;
    c2 = us[2] ^ Tx;
    if((c1|c2) & T2)
        goto bad;
    if(c0 >= T5) {
        if(n < 5)
            goto bad;
        c3 = us[3] ^ Tx;
        c4 = us[4] ^ Tx;
        if((c3|c4) & T2)
            goto bad;
        if(c0 >= T6) {
            /* 6 bytes */
            if(n < 6)
                goto bad;
            c5 = us[5] ^ Tx;
            if(c5 & T2)
                goto bad;
            wc = ((((((((((c0 & Mask6) << Bitx) |
                c1) << Bitx) | c2) << Bitx) |
                c3) << Bitx) | c4) << Bitx) | c5;
            if(wc <= Wchar5)
                goto bad;
            *p = wc;
            return 6;
        }
        /* 5 bytes */
        wc = (((((((c0 & Mask5) << Bitx) |
            c1) << Bitx) | c2) << Bitx) |
            c3) << Bitx) | c4;
        if(wc <= Wchar4)
            goto bad;
        *p = wc;
        return 5;
    }
    if(c0 >= T4) {
        /* 4 bytes */
        if(n < 4)
            goto bad;
        c3 = us[3] ^ Tx;
        if(c3 & T2)
            goto bad;
        wc = ((((((c0 & Mask4) << Bitx) |
            c1) << Bitx) | c2) << Bitx) |
            c3;
        if(wc <= Wchar3)
            goto bad;
        *p = wc;
        return 4;
    }
    /* 3 bytes */
    wc = ((((c0 & Mask3) << Bitx) |
        c1) << Bitx) | c2;
    if(wc <= Wchar2)
        goto bad;
    *p = wc;
    return 3;
}
if(c0 >= T2) {
    /* 2 bytes */
```

```
        if(n < 2)
            goto bad;
        c1 = us[1] ^ Tx;
        if(c1 & T2)
            goto bad;
        wc = ((c0 & Mask2) << Bitx) |
            c1;
        if(wc <= Wchar1)
            goto bad;
        *p = wc;
        return 2;
    }
    /* 1 byte */
    if(c0 >= Tx)
        goto bad;
    *p = c0;
    return 1;

bad:
    errno = EILSEQ;
    return -1;
}
```

⟨tcs/utf.c 296⟩≡

```
#ifdef PLAN9
#include    <u.h>
#include    <libc.h>
#include    <bio.h>
#else
#include    <sys/types.h>
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <unistd.h>
#include    <errno.h>
#include    "plan9.h"
#endif
#include    "hdr.h"

/*
    the our_* routines are implementations for the corresponding library
    routines. for a while, i tried to actually name them wctomb etc
    but stopped that after i found a system which made wchar_t an
    unsigned char.
*/

int our_wctomb(char *s, unsigned long wc);
int our_mbtowc(unsigned long *p, char *s, unsigned n);
int runetoisoutf(char *str, Rune *rune);
int fullisorune(char *str, int n);
int isochartorune(Rune *rune, char *str);
```

⟨*function* utf_in*(tcs)* 289e⟩

⟨*function* utf_out*(tcs)* 290a⟩

⟨*function* isoutf_in*(tcs)* 290b⟩

⟨*function* isoutf_out*(tcs)* 291a⟩

```
enum
{
    Char1   = Runeself, Rune1   = Runeself,
    Char21  = 0xA1,     Rune21  = 0x0100,
    Char22  = 0xF6,     Rune22  = 0x4016,
    Char3   = 0xFC,     Rune3   = 0x10000,  /* really 0x38E2E */
    Esc = 0xBE,     Bad = Runeerror
};

static  uchar   U[256];
static  uchar   T[256];
```

⟨*function* `mktable`*(tcs)* 291b⟩

⟨*function* `isochartorune`*(tcs)* 291c⟩

⟨*function* `runetoisoutf`*(tcs)* 292⟩

⟨*function* `fullisorune`*(tcs)* 293a⟩

```
#ifdef PLAN9
int errno;
#endif

enum
{
    T1  = 0x00,
    Tx  = 0x80,
    T2  = 0xC0,
    T3  = 0xE0,
    T4  = 0xF0,
    T5  = 0xF8,
    T6  = 0xFC,

    Bit1    = 7,
    Bitx    = 6,
    Bit2    = 5,
    Bit3    = 4,
    Bit4    = 3,
    Bit5    = 2,
    Bit6    = 2,

    Mask1   = (1<<Bit1)-1,
    Maskx   = (1<<Bitx)-1,
    Mask2   = (1<<Bit2)-1,
    Mask3   = (1<<Bit3)-1,
    Mask4   = (1<<Bit4)-1,
    Mask5   = (1<<Bit5)-1,
    Mask6   = (1<<Bit6)-1,

    Wchar1  = (1UL<<Bit1)-1,
    Wchar2  = (1UL<<(Bit2+Bitx))-1,
    Wchar3  = (1UL<<(Bit3+2*Bitx))-1,
    Wchar4  = (1UL<<(Bit4+3*Bitx))-1,
    Wchar5  = (1UL<<(Bit5+4*Bitx))-1,

#ifndef EILSEQ
    EILSEQ  = 123,
#endif /* EILSEQ */
```

```
};
```

⟨*function* `our_wctomb`*(tcs)* 293b⟩

⟨*function* `our_mbtowc`*(tcs)* 294⟩

tcs/tcs.c

⟨*function* `main`*(tcs)* 298⟩≡ (306)

```
int
main(int argc, char **argv)
{
    char *from = "utf";
    char *to = "utf";
    int fd;
    int listem = 0;
    struct convert *t, *f;

    ARGBEGIN {
    case 'c':
        clean = 1;
        break;
    case 'f':
        from = EARGF(usage());
        break;
    case 'l':
        listem = 1;
        break;
    case 's':
        squawk = 0;
        break;
    case 't':
        to = EARGF(usage());
        break;
    case 'v':
        verbose = 1;
        break;
    default:
        usage();
        break;
    } ARGEND

    USED(argc);
    if(verbose)
        squawk = 1;
    if(listem){
        list();
        EXIT(0, 0);
    }
    if(!from || !to)
        usage();
    f = conv(from, 1);
    t = conv(to, 0);
#define PROC    {if(f->flags&Table)\
            intable(fd, (long *)f->data, t);\
        else\
            ((Infn)(f->fn))(fd, (long *)0, t);}
    if(*argv){
        while(*argv){
```

```
            file = *argv;
#ifndef PLAN9
            if((fd = open(*argv, 0)) < 0){
                EPR "%s: %s: %s\n", argv0, *argv, strerror(errno));
#else /* PLAN9 */
            if((fd = open(*argv, OREAD)) < 0){
                EPR "%s: %s: %r\n", argv0, *argv);
#endif /* PLAN9 */
                EXIT(1, "open failure");
            }
            PROC
            close(fd);
            argv++;
        }
    } else {
        fd = 0;
        PROC
    }
    if(verbose)
        EPR "%s: %ld input bytes, %ld runes, %ld output bytes (%ld errors)\n", argv0,
            ninput, nrunes, noutput, nerrors);
    EXIT(((nerrors && squawk)? 1:0), ((nerrors && squawk)? "conversion error":0));
    return(0);  /* shut up compiler */
}
```

⟨*function* usage*(tcs)* 299a⟩≡ (306)

```
void
usage(void)
{
    EPR "Usage: %s [-slv] [-f cs] [-t cs] [file ...]\n", argv0);
    verbose = 1;
    list();
    EXIT(1, "usage");
}
```

⟨*function* list*(tcs)* 299b⟩≡ (306)

```
void
list(void)
{
    struct convert *c;
    char ch = verbose?'\t':' ';

#ifndef PLAN9
    EPR "%s version = '%s'\n", argv0, version);
#endif
    if(verbose)
        EPR "character sets:\n");
    else
        EPR "cs:");
    for(c = convert; c->name; c++){
        if((c->flags&From) && c[1].name && (strcmp(c[1].name, c->name) == 0)){
            EPR "%c%s", ch, c->name);
            c++;
        } else if(c->flags&Table)
            EPR "%c%s", ch, c->name);
        else if(c->flags&From)
            EPR "%c%s(from)", ch, c->name);
        else
            EPR "%c%s(to)", ch, c->name);
        if(verbose)
```

```
            EPR "\t%s\n", c->chatter);
    }
    if(!verbose)
        EPR "\n");
}
```

⟨*function* conv*(tcs)* 300a⟩≡ (306)

```
struct convert *
conv(char *name, int from)
{
    struct convert *c;

    for(c = convert; c->name; c++){
        if(cistrcmp(c->name, name) != 0)
            continue;
        if(c->flags&Table)
            return(c);
        if(((c->flags&From) == 0) == (from == 0))
            return(c);
    }
    EPR "%s: charset ‘%s’ unknown\n", argv0, name);
    EXIT(1, "unknown character set");
    return(0);  /* just shut the compiler up */
}
```

⟨*function* unicode_in*(tcs)* 300b⟩≡ (306)

```
void
unicode_in(int fd, long *notused, struct convert *out)
{
    Rune buf[N];
    int n;
    int swabme;

    USED(notused);
    if(read(fd, (char *)buf, 2) != 2)
        return;
    ninput += 2;
    switch(buf[0])
    {
    default:
        OUT(out, buf, 1);
    case 0xFEFF:
        swabme = 0;
        break;
    case 0xFFFE:
        swabme = 1;
        break;
    }
    while((n = read(fd, (char *)buf, 2*N)) > 0){
        ninput += n;
        if(swabme)
            swab2((char *)buf, n);
        if(n&1){
            if(squawk)
                EPR "%s: odd byte count in %s\n", argv0, file);
            nerrors++;
            if(clean)
                n--;
            else
                buf[n++/2] = Runeerror;
```

```
        }
        OUT(out, buf, n/2);
    }
    OUT(out, buf, 0);
}
```

⟨*function* `unicode_in_be`*(tcs)* 301a⟩≡ (306)

```
void
unicode_in_be(int fd, long *notused, struct convert *out)
{
    int i, n;
    Rune buf[N], r;
    uchar *p;

    USED(notused);
    while((n = read(fd, (char *)buf, 2*N)) > 0){
        ninput += n;
        p = (uchar*)buf;
        for(i=0; i<n/2; i++){
            r = *p++<<8;
            r |= *p++;
            buf[i] = r;
        }
        if(n&1){
            if(squawk)
                EPR "%s: odd byte count in %s\n", argv0, file);
            nerrors++;
            if(clean)
                n--;
            else
                buf[n++/2] = Runeerror;
        }
        OUT(out, buf, n/2);
    }
    OUT(out, buf, 0);
}
```

⟨*function* `unicode_in_le`*(tcs)* 301b⟩≡ (306)

```
void
unicode_in_le(int fd, long *notused, struct convert *out)
{
    int i, n;
    Rune buf[N], r;
    uchar *p;

    USED(notused);
    while((n = read(fd, (char *)buf, 2*N)) > 0){
        ninput += n;
        p = (uchar*)buf;
        for(i=0; i<n/2; i++){
            r = *p++;
            r |= *p++<<8;
            buf[i] = r;
        }
        if(n&1){
            if(squawk)
                EPR "%s: odd byte count in %s\n", argv0, file);
            nerrors++;
            if(clean)
                n--;
```

```
            else
                buf[n++/2] = Runeerror;
        }
        OUT(out, buf, n/2);
    }
    OUT(out, buf, 0);
}
```

⟨*function* `unicode_out`*(tcs)* 302a⟩≡ (306)

```
void
unicode_out(Rune *base, int n, long *notused)
{
    static int first = 1;

    USED(notused);
    nrunes += n;
    if(first){
        unsigned short x = 0xFEFF;
        noutput += 2;
        write(1, (char *)&x, 2);
        first = 0;
    }
    noutput += 2*n;
    write(1, (char *)base, 2*n);
}
```

⟨*function* `unicode_out_be`*(tcs)* 302b⟩≡ (306)

```
void
unicode_out_be(Rune *base, int n, long *notused)
{
    int i;
    uchar *p;
    Rune r;

    USED(notused);
    p = (uchar*)base;
    for(i=0; i<n; i++){
        r = base[i];
        *p++ = r>>8;
        *p++ = r;
    }
    nrunes += n;
    noutput += 2*n;
    write(1, (char *)base, 2*n);
}
```

⟨*function* `unicode_out_le`*(tcs)* 302c⟩≡ (306)

```
void
unicode_out_le(Rune *base, int n, long *notused)
{
    int i;
    uchar *p;
    Rune r;

    USED(notused);
    p = (uchar*)base;
    for(i=0; i<n; i++){
        r = base[i];
        *p++ = r;
        *p++ = r>>8;
```

```
    }
    nrunes += n;
    noutput += 2*n;
    write(1, (char *)base, 2*n);
}
```

⟨*function* `intable`*(tcs)* 303a⟩≡ (306)

```
void
intable(int fd, long *table, struct convert *out)
{
    uchar buf[N];
    uchar *p, *e;
    Rune *r;
    int n;
    long c;

    while((n = read(fd, (char *)buf, N)) > 0){
        ninput += n;
        r = runes;
        for(p = buf, e = buf+n; p < e; p++){
            c = table[*p];
            if(c < 0){
                if(squawk)
                    EPR "%s: bad char 0x%x near byte %ld in %s\n", argv0, *p, ninput+(p-e), file);
                nerrors++;
                if(clean)
                    continue;
                c = BADMAP;
            }
            *r++ = c;
        }
        OUT(out, runes, r-runes);
    }
    OUT(out, runes, 0);
    if(n < 0){
#ifdef  PLAN9
        EPR "%s: input read: %r\n", argv0);
#else
        EPR "%s: input read: %s\n", argv0, strerror(errno));
#endif
        EXIT(1, "input read error");
    }
}
```

⟨*function* `outtable`*(tcs)* 303b⟩≡ (306)

```
void
outtable(Rune *base, int n, long *map)
{
    long c;
    char *p;
    int i;

    nrunes += n;
    for(i = 0; i < NRUNE; i++)
        tab[i] = -1;
    for(i = 0; i < 256; i++)
        if(map[i] >= 0)
            tab[map[i]] = i;
    for(i = 0, p = obuf; i < n; i++){
        c = tab[base[i]];
```

```
        if(c < 0){
            if(squawk)
                EPR "%s: rune 0x%x not in output cs\n", argv0, base[i]);
            nerrors++;
            if(clean)
                continue;
            c = BADMAP;
        }
        *p++ = c;
    }
    noutput += p-obuf;
    write(1, obuf, p-obuf);
}
```

⟨*global* tabascii *(tcs)* 304a⟩≡ (306)

```
long tabascii[256] =
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
};
```

⟨*global* convert *(tcs)* 304b⟩≡ (306)

```
struct convert convert[] =
{   /* if two entries have the same name, put the from one first */
    { "8859-1", "Latin-1 (Western and Northern Europe including Italian)", Table, (void *)tab8859_1 },
    { "8859-2", "Latin-2 (Eastern Europe except Turkey and the Baltic countries)", Table, (void *)tab8859_2 },
    { "8859-3", "Latin-3 (Mediterranean, South Africa, Esperanto)", Table, (void *)tab8859_3 },
    { "8859-4", "Latin-4 (Scandinavia and the Baltic countries; obsolete)", Table, (void *)tab8859_4 },
    { "8859-5", "Part 5 (Cyrillic)", Table, (void *)tab8859_5 },
    { "8859-6", "Part 6 (Arabic)", Table, (void *)tab8859_6 },
    { "8859-7", "Part 7 (Greek)", Table, (void *)tab8859_7 },
    { "8859-8", "Part 8 (Hebrew)", Table, (void *)tab8859_8 },
    { "8859-9", "Latin-5 (Turkey, Western Europe except Icelandic and Faroese)", Table, (void *)tab8859_9 },
    { "8859-10", "Latin-6 (Northern Europe)", Table, (void *)tab8859_10 },
    { "8859-15", "Latin-9 (Western Europe)", Table, (void *)tab8859_15 },
    { "ascii", "7-bit ASCII", Table, (void *)tabascii },
    { "atari", "ATARI-ST character set", Table, (void *)tabatari },
    { "av", "Alternativnyj Variant", Table, (void *)tabav },
    { "big5", "Big 5 (HKU)", From|Func, 0, (Fnptr)big5_in },
    { "big5", "Big 5 (HKU)", Func, 0, (Fnptr)big5_out },
    { "ebcdic", "EBCDIC", Table, (void *)tabebcdic },   /* 6f is recommended bad map */
    { "euc-k", "Korean EUC: ASCII+KS C 5601 1987", From|Func, 0, (Fnptr)uksc_in },
    { "euc-k", "Korean EUC: ASCII+KS C 5601 1987", Func, 0, (Fnptr)uksc_out },
    { "euc-kr", "Korean EUC: ASCII+KS C 5601 1987", From|Func, 0, (Fnptr)uksc_in },
    { "euc-kr", "Korean EUC: ASCII+KS C 5601 1987", Func, 0, (Fnptr)uksc_out },
    { "ks_c_5601-1987", "Korean EUC: ASCII+KS C 5601 1987", From|Func, 0, (Fnptr)uksc_in },
```

```
{ "ks_c_5601-1987", "Korean EUC: ASCII+KS C 5601 1987", Func, 0, (Fnptr)uksc_out },
{ "gb2312", "GB2312-80 (Chinese)", From|Func, 0, (Fnptr)gb_in },
{ "gb2312", "GB2312-80 (Chinese)", Func, 0, (Fnptr)gb_out },
{ "gbk", "GBK (Chinese)", From|Func, 0, (Fnptr)gbk_in },
{ "gbk", "GBK (Chinese)", Func, 0, (Fnptr)gbk_out },
{ "html", "HTML", From|Func, 0, (Fnptr)html_in },
{ "html", "HTML", Func, 0, (Fnptr)html_out },
{ "ibm437", "IBM Code Page 437 (US)", Table, (void*)tabcp437 },
{ "ibm720", "IBM Code Page 720 (Arabic)", Table, (void*)tabcp720 },
{ "ibm737", "IBM Code Page 737 (Greek)", Table, (void*)tabcp737 },
{ "ibm775", "IBM Code Page 775 (Baltic)", Table, (void*)tabcp775 },
{ "ibm850", "IBM Code Page 850 (Multilingual Latin I)", Table, (void*)tabcp850 },
{ "ibm852", "IBM Code Page 852 (Latin II)", Table, (void*)tabcp852 },
{ "ibm855", "IBM Code Page 855 (Cyrillic)", Table, (void*)tabcp855 },
{ "ibm857", "IBM Code Page 857 (Turkish)", Table, (void*)tabcp857 },
{ "ibm858", "IBM Code Page 858 (Multilingual Latin I+Euro)", Table, (void*)tabcp858 },
{ "ibm862", "IBM Code Page 862 (Hebrew)", Table, (void*)tabcp862 },
{ "ibm866", "IBM Code Page 866 (Russian)", Table, (void*)tabcp866 },
{ "ibm874", "IBM Code Page 874 (Thai)", Table, (void*)tabcp874 },
{ "iso-2022-jp", "alias for jis-kanji (MIME)", From|Func, 0, (Fnptr)jisjis_in },
{ "iso-2022-jp", "alias for jis-kanji (MIME)", Func, 0, (Fnptr)jisjis_out },
{ "iso-8859-1", "alias for 8859-1 (MIME)", Table, (void *)tab8859_1 },
{ "iso-8859-2", "alias for 8859-2 (MIME)", Table, (void *)tab8859_2 },
{ "iso-8859-3", "alias for 8859-3 (MIME)", Table, (void *)tab8859_3 },
{ "iso-8859-4", "alias for 8859-4 (MIME)", Table, (void *)tab8859_4 },
{ "iso-8859-5", "alias for 8859-5 (MIME)", Table, (void *)tab8859_5 },
{ "iso-8859-6", "alias for 8859-6 (MIME)", Table, (void *)tab8859_6 },
{ "iso-8859-7", "alias for 8859-7 (MIME)", Table, (void *)tab8859_7 },
{ "iso-8859-8", "alias for 8859-8 (MIME)", Table, (void *)tab8859_8 },
{ "iso-8859-9", "alias for 8859-9 (MIME)", Table, (void *)tab8859_9 },
{ "iso-8859-10", "alias for 8859-10 (MIME)", Table, (void *)tab8859_10 },
{ "iso-8859-15", "alias for 8859-15 (MIME)", Table, (void *)tab8859_15 },
{ "jis", "guesses at the JIS encoding", From|Func, 0, (Fnptr)jis_in },
{ "jis-kanji", "ISO 2022-JP (Japanese)", From|Func, 0, (Fnptr)jisjis_in },
{ "jis-kanji", "ISO 2022-JP (Japanese)", Func, 0, (Fnptr)jisjis_out },
{ "koi8", "KOI-8 (GOST 19769-74)", Table, (void *)tabkoi8 },
{ "koi8-r", "alias for koi8 (MIME)", Table, (void *)tabkoi8 },
{ "latin1", "alias for 8859-1", Table, (void *)tab8859_1 },
{ "macrom", "Macintosh Standard Roman character set", Table, (void *)tabmacroman },
{ "microsoft", "alias for windows1252", Table, (void *)tabcp1252 },
{ "ms-kanji", "Microsoft, or Shift-JIS", From|Func, 0, (Fnptr)msjis_in },
{ "ms-kanji", "Microsoft, or Shift-JIS", Func, 0, (Fnptr)msjis_out },
{ "msdos", "IBM PC (alias for ibm437)", Table, (void *)tabcp437 },
{ "msdos2", "IBM PC (ibm437 with graphics in C0)", Table, (void *)tabmsdos2 },
{ "next", "NEXTSTEP character set", Table, (void *)tabnextstep },
{ "ov", "Osnovnoj Variant", Table, (void *)tabov },
{ "ps2", "IBM PS/2: (alias for ibm850)", Table, (void *)tabcp850 },
{ "sf1", "ISO-646: Finnish/Swedish SF-1 variant", Table, (void *)tabsf1 },
{ "sf2", "ISO-646: Finnish/Swedish SF-2 variant (recommended)", Table, (void *)tabsf2 },
{ "tis-620", "Thai+ASCII (TIS 620-1986)", Table, (void *)tabtis620 },
{ "tune", "TUNE (Tamil)", From|Func, 0, (Fnptr)tune_in },
{ "tune", "TUNE (Tamil)", Func, 0, (Fnptr)tune_out },
{ "ucode", "Russian U-code", Table, (void *)tabucode },
{ "ujis", "EUC-JX: JIS 0208", From|Func, 0, (Fnptr)ujis_in },
{ "ujis", "EUC-JX: JIS 0208", Func, 0, (Fnptr)ujis_out },
{ "unicode", "Unicode 1.1", From|Func, 0, (Fnptr)unicode_in },
{ "unicode", "Unicode 1.1", Func, 0, (Fnptr)unicode_out },
{ "unicode-be", "Unicode 1.1 big-endian", From|Func, 0, (Fnptr)unicode_in_be },
{ "unicode-be", "Unicode 1.1 big-endian", Func, 0, (Fnptr)unicode_out_be },
{ "unicode-le", "Unicode 1.1 little-endian", From|Func, 0, (Fnptr)unicode_in_le },
```

```
    { "unicode-le", "Unicode 1.1 little-endian", Func, 0, (Fnptr)unicode_out_le },
    { "us-ascii", "alias for ascii (MIME)", Table, (void *)tabascii },
    { "utf", "FSS-UTF a.k.a. UTF-8", From|Func, 0, (Fnptr)utf_in },
    { "utf", "FSS-UTF a.k.a. UTF-8", Func, 0, (Fnptr)utf_out },
    { "utf1", "UTF-1 (ISO 10646 Annex A)", From|Func, 0, (Fnptr)isoutf_in },
    { "utf1", "UTF-1 (ISO 10646 Annex A)", Func, 0, (Fnptr)isoutf_out },
    { "utf-8", "alias for utf (MIME)", From|Func, 0, (Fnptr)utf_in },
    { "utf-8", "alias for utf (MIME)", Func, 0, (Fnptr)utf_out },
    { "utf-16", "alias for unicode (MIME)", From|Func, 0, (Fnptr)unicode_in },
    { "utf-16", "alias for unicode (MIME)", Func, 0, (Fnptr)unicode_out },
    { "utf-16be", "alias for unicode-be (MIME)", From|Func, 0, (Fnptr)unicode_in_be },
    { "utf-16be", "alias for unicode-be (MIME)", Func, 0, (Fnptr)unicode_out_be },
    { "utf-16le", "alias for unicode-le (MIME)", From|Func, 0, (Fnptr)unicode_in_le },
    { "utf-16le", "alias for unicode-le (MIME)", Func, 0, (Fnptr)unicode_out_le },
    { "viet1", "Vietnamese VSCII-1 (1993)", Table, (void *)tabviet1 },
    { "viet2", "Vietnamese VSCII-2 (1993)", Table, (void *)tabviet2 },
    { "vscii", "Vietnamese VISCII 1.1 (1992)", Table, (void *)tabviscii },
    { "windows-1250", "Windows Code Page 1250 (Central Europe)", Table, (void *)tabcp1250 },
    { "windows-1251", "Windows Code Page 1251 (Cyrillic)", Table, (void *)tabcp1251 },
    { "windows-1252", "Windows Code Page 1252 (Latin I)", Table, (void *)tabcp1252 },
    { "windows-1253", "Windows Code Page 1253 (Greek)", Table, (void *)tabcp1253 },
    { "windows-1254", "Windows Code Page 1254 (Turkish)", Table, (void *)tabcp1254 },
    { "windows-1255", "Windows Code Page 1255 (Hebrew)", Table, (void *)tabcp1255 },
    { "windows-1256", "Windows Code Page 1256 (Arabic)", Table, (void *)tabcp1256 },
    { "windows-1257", "Windows Code Page 1257 (Baltic)", Table, (void *)tabcp1257 },
    { "windows-1258", "Windows Code Page 1258 (Vietnam)", Table, (void *)tabcp1258 },
    { 0 },
};
```

⟨tcs/tcs.c 306⟩≡

```
#ifndef PLAN9
#include    <sys/types.h>
#include    <stdio.h>
#include    <unistd.h>
#include    <stdlib.h>
#include    <fcntl.h>
#include    <string.h>
#include    <errno.h>
#include    "plan9.h"
#else /* PLAN9 */
#include    <u.h>
#include    <libc.h>
#include    <bio.h>
#endif /* PLAN9 */
#include    "cyrillic.h"
#include    "misc.h"
#include    "ms.h"
#include    "8859.h"
#include    "big5.h"
#include    "gb.h"
#include    "hdr.h"
#include    "conv.h"

void usage(void);
void list(void);
int squawk = 1;
int clean = 0;
int verbose = 0;
long ninput, noutput, nrunes, nerrors;
char *file = "stdin";
```

```
char *argv0;
Rune runes[N];
char obuf[UTFmax*N];    /* maximum bloat from N runes */
long tab[NRUNE];
#ifndef PLAN9
extern char version[];
#endif

void intable(int, long *, struct convert *);
void unicode_in(int, long *, struct convert *);
void unicode_out(Rune *, int, long *);
```

⟨*function* `main` *(tcs)* 298⟩
⟨*function* `usage` *(tcs)* 299a⟩

⟨*function* `list` *(tcs)* 299b⟩
⟨*function* `conv` *(tcs)* 300a⟩

```
void
swab2(char *b, int n)
{
    char *e, p;

    for(e = b+n; b < e; b++){
        p = *b;
        *b = b[1];
        *++b = p;
    }
}
```

⟨*function* `unicode_in` *(tcs)* 300b⟩

⟨*function* `unicode_in_be` *(tcs)* 301a⟩
⟨*function* `unicode_in_le` *(tcs)* 301b⟩

⟨*function* `unicode_out` *(tcs)* 302a⟩

⟨*function* `unicode_out_be` *(tcs)* 302b⟩
⟨*function* `unicode_out_le` *(tcs)* 302c⟩

⟨*function* `intable` *(tcs)* 303a⟩
⟨*function* `outtable` *(tcs)* 303b⟩

⟨*global* `tabascii` *(tcs)* 304a⟩

```
long tabmsdos[256] =    /* from jhelling@cs.ruu.nl (Jeroen Hellingman) */
{
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x00c7, 0x00fc, 0x00e9, 0x00e2, 0x00e4, 0x00e0, 0x00e5, 0x00e7, /* latin */
0x00ea, 0x00eb, 0x00e8, 0x00ef, 0x00ee, 0x00ec, 0x00c4, 0x00c5,
0x00c9, 0x00e6, 0x00c6, 0x00f4, 0x00f6, 0x00f2, 0x00fb, 0x00f9,
```

```
0x00ff, 0x00d6, 0x00dc, 0x00a2, 0x00a3, 0x00a5, 0x20a7, 0x0192,
0x00e1, 0x00ed, 0x00f3, 0x00fa, 0x00f1, 0x00d1, 0x00aa, 0x00ba,
0x00bf, 0x2310, 0x00ac, 0x00bd, 0x00bc, 0x00a1, 0x00ab, 0x00bb,
0x2591, 0x2592, 0x2593, 0x2502, 0x2524, 0x2561, 0x2562, 0x2556, /* forms */
0x2555, 0x2563, 0x2551, 0x2557, 0x255d, 0x255c, 0x255b, 0x2510,
0x2514, 0x2534, 0x252c, 0x251c, 0x2500, 0x253c, 0x255e, 0x255f,
0x255a, 0x2554, 0x2569, 0x2566, 0x2560, 0x2550, 0x256c, 0x2567,
0x2568, 0x2564, 0x2565, 0x2559, 0x2558, 0x2552, 0x2553, 0x256b,
0x256a, 0x2518, 0x250c, 0x2588, 0x2584, 0x258c, 0x2590, 0x2580,
0x03b1, 0x00df, 0x0393, 0x03c0, 0x03a3, 0x03c3, 0x00b5, 0x03c4, /* greek */
0x03a6, 0x0398, 0x2126, 0x03b4, 0x221e, 0x2205, 0x2208, 0x2229,
0x2261, 0x00b1, 0x2265, 0x2264, 0x2320, 0x2321, 0x00f7, 0x2248, /* math */
0x00b0, 0x2022, 0x00b7, 0x221a, 0x207f, 0x00b2, 0x220e, 0x00a0,
};
long tabmsdos2[256] =   /* from jhelling@cs.ruu.nl (Jeroen Hellingman) */
{
0x0000, 0x263a, 0x263b, 0x2665, 0x2666, 0x2663, 0x2660, 0x2022,
0x25d8, 0x25cb, 0x25d9, 0x2642, 0x2640, 0x266a, 0x266b, 0x263c,
0x25b6, 0x25c0, 0x2195, 0x203c, 0x00b6, 0x00a7, 0x2043, 0x21a8,
0x2191, 0x2193, 0x2192, 0x2190, 0x2319, 0x2194, 0x25b2, 0x25bc,
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,
0x00c7, 0x00fc, 0x00e9, 0x00e2, 0x00e4, 0x00e0, 0x00e5, 0x00e7, /* latin */
0x00ea, 0x00eb, 0x00e8, 0x00ef, 0x00ee, 0x00ec, 0x00c4, 0x00c5,
0x00c9, 0x00e6, 0x00c6, 0x00f4, 0x00f6, 0x00f2, 0x00fb, 0x00f9,
0x00ff, 0x00d6, 0x00dc, 0x00a2, 0x00a3, 0x00a5, 0x20a7, 0x0192,
0x00e1, 0x00ed, 0x00f3, 0x00fa, 0x00f1, 0x00d1, 0x00aa, 0x00ba,
0x00bf, 0x2310, 0x00ac, 0x00bd, 0x00bc, 0x00a1, 0x00ab, 0x00bb,
0x2591, 0x2592, 0x2593, 0x2502, 0x2524, 0x2561, 0x2562, 0x2556, /* forms */
0x2555, 0x2563, 0x2551, 0x2557, 0x255d, 0x255c, 0x255b, 0x2510,
0x2514, 0x2534, 0x252c, 0x251c, 0x2500, 0x253c, 0x255e, 0x255f,
0x255a, 0x2554, 0x2569, 0x2566, 0x2560, 0x2550, 0x256c, 0x2567,
0x2568, 0x2564, 0x2565, 0x2559, 0x2558, 0x2552, 0x2553, 0x256b,
0x256a, 0x2518, 0x250c, 0x2588, 0x2584, 0x258c, 0x2590, 0x2580,
0x03b1, 0x00df, 0x0393, 0x03c0, 0x03a3, 0x03c3, 0x00b5, 0x03c4, /* greek */
0x03a6, 0x0398, 0x2126, 0x03b4, 0x221e, 0x2205, 0x2208, 0x2229,
0x2261, 0x00b1, 0x2265, 0x2264, 0x2320, 0x2321, 0x00f7, 0x2248, /* math */
0x00b0, 0x2022, 0x00b7, 0x221a, 0x207f, 0x00b2, 0x220e, 0x00a0,
};
```

⟨*global* `convert`*(tcs)* 304b⟩

Glossary

```
URL  = Uniform Resource Locator
URI  = Uniform Resource Identifier
HTML = HyperText Markup Language
HTTP = HyperText Transfer Protocol
HTTPS= HTTP over TLS
TLS  = Transport Layer Security
DOM  = Document Object Model
MIME = Multi-Purpose Internet Mail Extensions
WWW  = World Wide Web
CSS  = Cascading Style Sheets
JS   = JavaScript
9P   = The Plan 9 file-system protocol
CLI  = Command-Line Interface
GUI  = Graphical User Interface
```

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) 10

[Pad09] Yoann Padioleau. Syncweb, literate programming meets unison, 2009. https://github.com/aryx/syncweb. cited page(s) 10

[Pad14] Yoann Padioleau. *Principia Softwarica: The Plan 9 Kernel 9*. 2014. cited page(s) 7, 9, 13, 39, 40

[Pad16a] Yoann Padioleau. *Principia Softwarica: The Plan 9 Core Libraries*. 2016. cited page(s) 7, 10, 39, 40

[Pad16b] Yoann Padioleau. *Principia Softwarica: The Plan 9 Network Stack /net*. 2016. cited page(s) 7, 9, 40

[Pad16c] Yoann Padioleau. *Principia Softwarica: The Plan 9 Windowing System rio*. 2016. cited page(s) 7, 8, 9, 40

[Pad18] Yoann Padioleau. *Principia Softwarica: The Plan 9 Shell rc*. 2018. cited page(s) 39, 40

[Pad26] Yoann Padioleau. *Principia Softwarica: The Plan 9 Widgets libpanel*. 2026. cited page(s) 7, 10, 40, 41