# The ELinks Manual

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>The ELinks Manual | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | May 4, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# List of Tables

# 1 Preface

Welcome! This is the entry point for the humble ELinks manual. It is by no means complete, it is not even very homogeneous and it should eventually be superseded by a complete ELinks Book. Until this happens you may also find it necessary to refer to the manual page for a very quick reference, or the built-in documentation available via the --long-help and --config-help ELinks command-line arguments. The built-in documentation is sure to be up-to-date.

There was a complete (or, for the most part complete) manual for Links 0.82 at one time, and you can still find it at:

- http://links.sourceforge.net/docs/manual-0.82-en/index.html

While large parts of it do not apply anymore, you may still find some relevant information there.

Authors:


- Jonas Fonseca <fonseca@diku.dk>
- Jose Luis Gonzalez Gonzalez <jlg80@mi.madritel.es>
- Laurent Monin <zas@norz.org>
- Miciah Dashiel Butler Masters <miciah@myrealbox.com>
- Petr Baudis <pasky@ucw.cz>
- Peter Wang


and others.



# 2 Getting ELinks up and running

Installing ELinks should be pretty easy on most systems. Below is described the whole process of configuring the compilation, compiling and installing.

## 2.1 Building and Installing ELinks

The quick guide for the impatient. Issue the following commands from the ELinks source directory:

```
$ ./configure && make && make install
```

However you might consider to tweek a few compile time options before building ELinks. You might also want to take a look at what libraries and programs are required or recommended that you install.

## 2.2 Requirements

To successfully install ELinks all that is required is GNU make (version 3.78 or later) and a C compiler. If you want to install directly from GIT it is also required that automake and autoconf is installed on your system.

## 2.3 Recommended Libraries and Programs

To make use of many of ELinks features you will need to have some external libraries and utility programs installed on your system. It is recommended to have the following libraries and programs installed:

| Dependency | Description |
|---|---|
| Lua or Guile | Makes it possible to write scripting plugins. See the Elinks - Lua interface for more info. |
| zlib 1.2.0.2 or later | For handling gzip or deflate compressed documents both locally and sent from server. |
| bzip2 | Likewise, for bzip2 compressed documents. |
| LZMA Utils | Likewise, for LZMA compressed documents. Version 4.32.5 should work. XZ Utils does not work. |
| OpenSSL, GNU TLS, or nss_compat_ossl | For handling secure HTTP browsing. |
| pkg-config | Needed for locating some libraries (at least GNU TLS, TRE, and SpiderMonkey) |
| GPM | *General Purpose Mouse* for mouse support. |
| expat | *XML Parser Toolkit* needed for XBEL support. |
| TRE | For regexp searching. Version 0.8.0 works. |
| libsmbclient | Library needed for smb:// protocol support. |
| rxvt-unicode | For terminal emulator which supports 88 colors. |
| xterm with 256 colors | Program atleast patch level 179 or rxvt program from version 2.7.9 for support of 256 colors. Newer PuTTY also has 256 color support. |
| libidn | For internationalized domain names. |
| SpiderMonkey | Mozilla's JavaScript engine for getting JavaScript/ECMAScript support in ELinks. See also notes on ECMAScript support. |

When compiling, you also need the header files for the libraries. Most OS distributors put a given library's headers in a package separate from the library itself; this package usually has `-dev` or similar appended to its name.

## 2.4 Further reading

This installation guide is far from being complete. You are also advised to read the `README` and the `INSTALL` files distributed with ELinks for further up to date info on building and installing.

## 2.5 Tips to obtain a very small static elinks binary

Tips to obtain a very small static elinks binary suitable for mini distributions

Remove config.cache (previous CC may be cached):

```
$ rm config.cache
```

Use dietlibc (http://www.fefe.de/dietlibc/) or similar stuff (uClibc, . . . ):

```
$ export CC='diet -Os gcc'
```

Use compilers flags to optimize for size:

```
$ export CFLAGS='-s -fno-inline -nostdinc -fomit-frame-pointer'
```

Note that if you don't use dietlibc, you definitively want to add `-Os` or `-O2` to `CFLAGS`; GCC 2.95 does not know `-Os`, and some say `-O2` gives smaller executables even for GCC 3.x.

**Tip**
If you use these `CFLAGS` on Cygwin and you get unresolved symbols (`htons` and suite in particular), try removing `-fno-inline` parameter.

Disable some compile-time options:

```
$ ./configure --disable-ipv6 --disable-backtrace --disable-nls \
  --enable-fastmem --without-zlib --without-bzlib --disable-xbel \
  --without-lua --without-gnutls --without-openssl --without-x \
  --enable-small --without-spidermonkey --without-gpm
```

You can disable bookmarks, globhist and more, too, if you want to.

**Other configure options that can reduce the size**

- --disable-backtrace disables internal backtrace code.
- --disable-nls disables i18n support.
- --enable-fastmem disables internal `malloc()` debugging and use `alloca()` wherever possible.
- --enable-small forces to remove some text descriptions in options and keybind stuff (regain 30Kb).

Dependencies over external libs must be removed using the related configure options:

| Option | Description |
|---|---|
| --without-zlib | removes libz dependency (compression) |
| --without-bzlib | removes libbz2 dependency (compression) |
| --disable-xbel | removes expat dependency (XBEL bookmarks support) |
| --without-lua | removes liblua dependency (Lua scripting) |
| --without-gnutls | removes libtls dependency (SSL support) |
| --without-openssl | removes libssl dependency (SSL support) |
| --without-x | removes libx11 dependency (restoring terminal title) |
| --without-spidermonkey | removes libjs dependency (JavaScript) |
| --without-gpm | removes libgpm dependency (mouse/console) |

It seems GCC 2.95.x do not generate as small binaries as GCC 3.2.x with same flags.

You can use an executable compressor like UPX http://upx.sourceforge.net/.

Here are some results using gcc 2.95.3, dietlibc-0.23, and previous flags:

```
me$ ls -l src/elinks
-rwxr-xr-x   1 zas      users      495100 Oct 20 15:53 src/elinks
me$ upx --best src/elinks
me$ ls -l src/elinks
-rwxr-xr-x   1 zas      users      217946 Oct 20 15:53 src/elinks
```

Whow ! Around 200kb :)

**Details about the --enable-small configure option effects:**

- it disables long descriptions of options;
- it disables textual descriptions of keybinding options;
- it reduces size of some HTTP errors messages;
- it disables fastfind feature, reducing performance, but also reducing a lot memory usage, and a bit the executable size.

## 2.6 ECMAScript support?!

Yes, there is some ECMAScript support in ELinks. There isn't anything we could call complete, but some bits could help with the most common ECMAScript usage cases - help you (and then us ;) get into your banking account, pass through those ignorant page redirects done by JavaScript code snippets and so.

ELinks does not have own ECMAScript parser and compiler; instead it reuses other people's work (this may eventually change, see the bottom of this file). First we aimed at the NJS engine, which is easy to install, small and compact; has nice naming scheme, horrible calling conventions and very lacking documentation; is not actively developed; and generally looks broken and extremely clumsy to work with. So we instead went the way of the SpiderMonkey (SM) engine (property of Mozilla), which is hard to install, bigger (mind you, it comes from Mozilla ;), has ugly naming scheme but nice calling conventions, acceptable documentation, is actively developed and ought to work.

### 2.6.1 Ok, so how to get the ECMAScript support working?

On Debian testing (Etch) or unstable (SID), run the following:

```
$ apt-get install libmozjs-dev
```

On Debian stable (Sarge), run the following:

```
$ apt-get install libsmjs-dev
```

On Arch Linux, run the following:

```
$ pacman -S js78
```

SpiderMonkey is disabled by default, enable it like this:

```
$ meson build -Dspidermonkey=true
```

Check for the following line in the features summary:

```
Run-time dependency mozjs-78 found: YES 78.15.0
```

Then run:

```
$ cd build/
$ ninja
$ sudo ninja install
```

Enjoy.

### 2.6.2 The ECMAScript support is buggy! Shall I blame Mozilla people?

Likely not. The ECMAScript engine provides only the language compiler and some basic built-in objects, and it's more than likely that the problem is on our side in the implementation of some of the HTML/DOM objects (perhaps we just haven't bothered to implement it at all yet). So better tell us first, and if we think it's not our fault we will tell you to go complain to Mozilla (better yet if it does not work in the Mozilla browsers neither ;-).

### 2.6.3 Now, I would still like NJS or a new JS engine from scratch...

...and you don't fear some coding? That's fine then! ELinks is in no way tied to SpiderMonkey, in fact the ECMAScript support was carefully implemented so that there are no SpiderMonkey references outside of `src/ecmascript/spidermonkey.*`. If you want to implement an alternative ECMAScript backend, go ahead - you will just need to write an autoconf detection for it and tie it to `src/ecmascript/ecmascript.c`, which should be easy. We await your patches eagerly.

## 2.7 Feature configuration file (`features.conf`)

This file contains various compile-time configuration settings, which you can adjust below. You can fine-tune the ELinks binary to include really only what you want it to. It acts as a front-end to the configure script in the sense that it is possible to control any features in this file by passing arguments to the configure script. In fact any arguments given to the script will overrule the values set in this file.

There are still some things which are to be adjusted only directly through the configure script arguments though, so check ./configure --help out as well!

All dependency checking is done by the configure script so even though a feature is enabled here it is possible that it will be disabled at compile time if the dependencies are not met. Check the features.log file generated by the configure script to make sure.

### 2.7.1 Notes for users

All features that can be controlled using this file are already set to their default values. The syntax used is hopefully familiar to most people.

# chars start a comment that runs until the end of the line.

The features are controlled by setting the various CONFIG_<FEATURE> variables to either *yes* or *no* depending on whether it should be enabled or disabled. So in order to disable bookmark support a line in this file should say:

```
CONFIG_BOOKMARKS=no
```

It is also possible to simply comment out the line in order to disable it. Therefore, if the default doesn't suit you, you can either comment it out or set it to the value you desire.

### 2.7.2 Bookmarks (`CONFIG_BOOKMARKS`)

ELinks has built-in hierarchic bookmarks support. Open the bookmarks manager by pressing *s*. When bookmarks are enabled, also support for the internal ELinks bookmarks format is always compiled in.

This is a favourite target for disabling in various embedded applications. It all depends on your requirements.

Also read the "The Ultimate Bookmarks Guide" in doc/bookmarks.txt

**Default:** enabled

### 2.7.3 XBEL Bookmarks (`CONFIG_XBEL_BOOKMARKS`)

ELinks also supports universal XML bookmarks format called XBEL, also supported by e.g. Galeon, various "always-have-my-bookmarks" websites and number of universal bookmark converters.

Frequently, you know you will not need it, then you can of course happily forcibly remove support for it and save few bytes.

**Default:** enabled if libexpat is found and bookmarks are enabled

### 2.7.4 Cookies (`CONFIG_COOKIES`)

Support for HTTP cookies --- a data token which the server sends the client once and then the client sends it back along each request to the server. This mechanism is crucial e.g. for keeping HTTP sessions (you "log in" to a site, and from then on the site recognizes you usually because of the cookie), but also for various banner systems, remembering values filled to various forms, and so on. You can further tune the ELinks behaviour at runtime (whether to accept/send cookies, ask for confirmation when accepting a cookie etc).

This functionality is usually quite important and you should not disable it unless you really know what are you doing.

**Default:** enabled

#### 2.7.5 Form History (CONFIG_FORMHIST)

The famous Competing Browser has that annoying thing which pops up when you submit a form, offering to remember it and pre-fill it the next time. And yes, ELinks can do that too! You will still need to also enable this manually at document.browse.forms.show_formhist.

Many people find it extremely annoying (including pasky), however some others consider it extremely handy and will sacrifice almost anything to get it. It will not do any harm to have this compiled-in as long as you will leave it turned off (which is also the default configuration).

**Default:** enabled

#### 2.7.6 Global History (CONFIG_GLOBHIST)

This device records each and every page you visit (to a configurable limit). You can browse through this history in the history manager (press *h*). Do not confuse this with the "session history", recording history of your browsing in the frame of one session (session history is the thing you move through when pressing *back* and *unback* or which you see in the File::History menu).

Global history does not care about the order you visited the pages in, it just records that you visited it, when did you do that and the title of the page. Then, you can see when did you visit a link last time (and what was the title of the target document at that time), links can be coloured as visited etc.

If you disable this feature, you will not lose any crucial functionality, just some relatively minor convenience features, which can nevertheless prove sometimes very practical.

**Default:** enabled

#### 2.7.7 MIME

ELinks uses a MIME system for determining the content type of documents and configuring programs for external handling. By default the option system can be used to configure how media types are handled. More info about how to set up the MIME handling using the option system can be found in the doc/mime.html file.

Below are listed some additional ways to do it.

#### 2.7.8 Mailcap (CONFIG_MAILCAP)

Mailcap files describe what program - on the local system - can be used to handle a media type. The file format is defined in RFC 1524 and more info including examples can be found in the doc/mailcap.html file.

This is very useful especially for clean interoperability with other MIME-aware applications and fitting nicely into the UNIX system, where this is the standard way of specifying MIME handlers. If you are not interested in that, you can still use the internal MIME associations system, though.

**Default:** enabled

#### 2.7.9 Mimetypes File (CONFIG_MIMETYPES)

Mimetypes file can be used to specify the relation between media types and file extensions.

Basically same thing applies here as for the mailcap support.

**Default:** enabled

#### 2.7.10 Gzip and Deflate Decompression (CONFIG_GZIP)

This makes ELinks send "Accept-Encoding: deflate, gzip" in HTTP requests and decompress any documents received in those formats. It works with local *.gz files as well.

**Default:** enabled if zlib is installed and new enough

#### 2.7.11 Bzip2 Decompression (CONFIG_BZIP2)

This makes ELinks decompress local *.bz2 files. Also, ELinks sends "Accept-Encoding: bzip2" in HTTP requests and decompresses any documents received in that format, but this encoding has not been registered at http://www.iana.org/assignments/-http-parameters, so most servers probably won't use it.

**Default:** enabled if the library is installed

#### 2.7.12 LZMA Decompression (CONFIG_LZMA)

This makes ELinks decompress local *.lzma files. Also, ELinks sends "Accept-Encoding: lzma" in HTTP requests and decompresses any documents received in that format, but this encoding has not been registered at http://www.iana.org/assignments/-http-parameters, so most servers probably won't use it.

To use this, first install LZMA Utils. Version 4.32.5 works; 4.42.2alpha also works and understands a newer LZMA file format. This version of ELinks does not support LZMA SDK from 7-Zip.

**Default:** disabled

#### 2.7.13 IPv6 Protocol Support (CONFIG_IPV6)

You know this thing that was designed to obsolete IPv4 but only pasky, weirdos and projects supported with big funds really use. ;-)

**Default:** enabled if the system supports it

#### 2.7.14 URI Rewriting (CONFIG_URI_REWRITE)

The goto dialog through which new URIs can be entered is an essential part of browsing in ELinks. This feature makes the dialog more powerful by making it possible to extend how entered text is handled through a set of rewrite rules (see protocol.rewrite options).

There are two types of rules: simple and smart ones.

Simple rewriting rules are basically URI abbreviations, making it possible to map a word to the full URI. They can also be used for hierarchic navigation to ease moving from some nested directory to the parent directory or doing other stuff with the current URI. For example, when you type *gg* into the goto dialog, you will be materialized at Google's homepage.

Smart rules can take arguments and therefore enable more advanced rewriting. The arguments could be search words to google for or a lookup query for a dictionary. Eg. type *gg:Petr Baudis king of ELinks cvs*.

This feature is also available in a more powerful form in the Lua and Guile extensions, so if you plan to or already use those, you won't miss anything by disabling this feature (besides easier and better integrated configuration).

**Default:** enabled

#### 2.7.15 BitTorrent Protocol Support (CONFIG_BITTORRENT)

The BitTorrent protocol is a protocol for distributing files in a peer-to-peer (P2P) manner. It uses the HTTP protocol for communicating with a central server and a peer-to-peer (P2P) protocol for exchanging file pieces betweens peer downloaders. The integrity of file pieces downloaded from peers are checked using cryptographic hashing (SHA1).

Downloads using BitTorrent are started by first downloading a .torrent file with the MIME type "application/x-bittorrent". The file contains information which enables ELinks to ask a central server, called a tracker, for information about other downloading peers and start downloading from and uploading to them.

At any time, an external handler can always be defined to take precedence of the internal BitTorrent client and the internal client can always be forced by prefixing the URI of the .torrent file with "bittorrent:"

**Note**
The BitTorrent support is still experimental.

**Default:** disabled

#### 2.7.16 Local CGI Support (`CONFIG_CGI`)

ELinks can (like w3m or lynx) execute certain executable files stored on the local disks as CGIs, when you target it on them (through a URI of the *file* scheme). ELinks emulates the complete CGI environment, like the program would be executed by a web server. See the protocol.file.cgi options tree for detailed runtime configuration.

Some people just write their bookmark management application as Perl CGI script and then access it from the web browser using this feature, not needing any web server or so. Therefore, this is a great possible way to extended the browser capabilities.

Even when you compile this in, you need to enable this yet in the configuration, and even then only CGI files passing certain user-defined filters (path-based) will be allowed to be executed (and there are certain other security barriers in place).

**Default:** disabled, available if setenv() or putenv() is found

#### 2.7.17 Data URI protocol (`CONFIG_DATA`)

The data URI protocol is defined in RFC 2397 and allows inclusion of small data items as "immediate" data, as if it had been included externally.

A data URL might be used for arbitrary types of data. The URI

```
data:,A%20brief%20note
```

encodes the text/plain string "A brief note", which might be useful in a footnote link.

**Default:** enabled

#### 2.7.18 Finger User Information Protocol Support (`CONFIG_FINGER`)

The finger protocol is a simple protocol defined in RFC 1288. The server return a friendly, human-oriented status report on either the system at the moment or a particular person in depth such as whether a user is currently logged-on, e-mail address, full name etc. As well as standard user information, it displays the contents of ".plan" file in the user's home directory. Often this file (maintained by the user) contained either useful information about the user's current activities, or alternatively all manner of humor.

It is most often implemented on Unix or Unix-like systems however due to security and privacy reasons it is usually disabled or only allowed locally on the system.

**Default:** disabled

#### 2.7.19 File Service Protocol (`CONFIG_FSP`)

File Service Protocol (FSP) is a very lightweight UDP based protocol for transferring files. FSP has many benefits over FTP, mainly for running anonymous archives. FSP protocol is valuable in all kinds of environments because it is one of the only TCP/IP protocols that is not aggressive about bandwidth, while still being sufficiently fault tolerant.

FSP is what anonymous FTP **should** be!

See http://fsp.sourceforge.net/ for more info.

**Default:** disabled

#### 2.7.20 File Transfer Protocol Support (CONFIG_FTP)

The File Transfer Protocol (FTP) is a software standard for transferring computer files between machines with widely different operating systems.

Many sites that run FTP servers enable so-called "anonymous ftp". Under this arrangement, users do not need an account on the server. By default, the account name for the anonymous access is *anonymous*. This account does not need a password, but users are commonly asked to send their email addresses as their passwords for authentication (protocol.ftp.anon_passwd), but there is no verification.

See also http://en.wikipedia.org/wiki/Ftp .

**Default:** enabled

#### 2.7.21 Gopher Protocol Support (CONFIG_GOPHER)

Gopher is a distributed document search and retrieval network protocol designed for the Internet in RFC 1436. The need for gopher arose in in the early days of the hypertext Internet where the number of documents that were being published in campus and research environments could not easily be distributed using known protocols like FTP because these documents were stored not in one place, but in many computers connected to the Internet.

The support works much like local file browsing with directories (aka. menus) and various file types that can be downloaded and viewed.

It is still very experimental and the CSO phone-book protocol is not implemented.

**Default:** disabled

#### 2.7.22 NNTP Protocol Support (CONFIG_NNTP)

Network news transport protocol support makes it possible to access nntp and news servers and read postings. It is still very experimental and is far from being considered a “news reader”.

It is possible to list news groups on a server, articles in a news group and retrieve articles by their number or message-id.

**Default:** disabled

#### 2.7.23 SMB Protocol Support (CONFIG_SMB)

ELinks supports browsing over the SMB protocol (URI *smb* scheme), using the libsmbclient library as back-end. Therefore, in order to have this enabled, you will need to install Samba (or at least just the libsmbclient part, if you can install it separately).

This use of libsmbclient is believed to be immune to the command injection attacks (CVE-2006-5925, bug 841) from which earlier ELinks releases (0.9.0 to 0.11.1) suffered.

**Default:** disabled

#### 2.7.24 Cascading Style Sheets (CONFIG_CSS)

Simplistic CSS support. It is still very much in it’s infancy so don’t expect too much. If you have use of background colors enabled more pages will have the intended background color. Also quite a few additional text attributes are applied. One example is highlighting of search words on Google’s cached pages.

There are options to disable both imported style sheets to minimize network traffic and whether to use CSS at all. Also a default style sheet can be defined to control the basic layout in the HTML renderer.

**Default:** enabled

#### 2.7.25 HTML Highlighting (CONFIG_HTML_HIGHLIGHT)

Makes it possible to view HTML source with the markup highlighted in colors configurable using CSS. It also makes values of referencing attributes accessible like the href="<uri>" attribute in <a> elements.

The HTML highlighting uses components of an experimental DOM implementation still in progress so enabling this feature will add a considerable amount of code to the compiled binary. On the other hand it will help to debug what will hopefully evolve into the next generation document renderer.

**Default:** disabled, requires that CSS is enabled

#### 2.7.26 ECMAScript (JavaScript) Browser Scripting (CONFIG_SCRIPTING_SPIDERMONKEY)

By enabling this feature, certain parts of ELinks, such as the goto URL dialog, may be extended using ECMAScript (aka. JavaScript) scripts. This can be useful to optimise your usage of ELinks.

For example you can define shortcuts (or abbreviations) for URLs of sites you often visit by having a goto URL hook expand them. This can also be achieved with the URI rewrite feature (CONFIG_URI_REWRITE), however it is not as powerful as doing it with scripting.

**Default:** enabled if Spidermonkey is found

#### 2.7.27 Mouse Support (CONFIG_MOUSE)

ELinks may be controlled not only by keyboard, but also by mouse to quite some extent. You can select links, menu items, scroll document, click at buttons etc, and it should hopefully work. ELinks supports mouse control by GPM, xterm mouse reporting and TWAIN's twterm mouse reporting.

It is generally nice convenience and doesn't cost too much. However, you can do everything with keyboard as you can with mouse. Also note that the xterm mouse reporting takes control over the terminal so that copy and pasting text from and to ELinks has to be done by holding down the Shift key.

**Default:** enabled

#### 2.7.28 88 Colors in Terminals (CONFIG_88_COLORS)

Define to add support for using 88 colors in terminals. Note that this requires a capable terminal emulator, such as:

- Thomas Dickey's XTerm, version 111 or later (check which version you have with xterm -version) compiled with --enable-88-color.
- Rxvt, version 2.7.9 or later compiled with --enable-88-color.

You will still need to enable this at runtime for a given terminal in terminal options, or set your $TERM variable to xterm-88color - then, ELinks will automatically configure itself to make use of all the available terminal features, while still acting sensibly when you happen to run it in an xterm w/o the 88 colors support.

When enabled, the memory usage is somewhat increased even when running in mono and 16 colors mode (the memory consumption can be especially remarkable when rendering very large documents and/or using very large terminals). However, when you actually run it in the suitable terminal, it looks really impressive, I'd say marvelous!

**Default:** disabled

#### 2.7.29 256 Colors in Terminals (CONFIG_256_COLORS)

Define to add support for using 256 colors in terminals. Note that this requires a capable terminal emulator, such as:

- Thomas Dickey's XTerm, version 111 or later (check which version you have with xterm -version) compiled with --enable-256-color.

- Rxvt, version 2.7.9 or later compiled with --enable-256-color.
- Recent versions of PuTTY also have some support for 256 colors.

You will still need to enable this at runtime for a given terminal in terminal options, or set your $TERM variable to xterm-256color - then, ELinks will automatically configure itself to make use of all the available terminal features, while still acting sensibly when you happen to run it in an xterm w/o the 256 colors support.

When enabled, the memory usage is somewhat increased even when running in mono and 16 colors mode (the memory consumption can be especially remarkable when rendering very large documents and/or using very large terminals). However, when you actually run it in the suitable terminal, it looks really impressive, I'd say marvelous!

**Default:** disabled

#### 2.7.30 True color (`CONFIG_TRUE_COLOR`)

Define to add support for True color. This mode eats a lot of memory.

**Default:** disabled

#### 2.7.31 Terminfo (`CONFIG_TERMINFO`)

Whether to use terminfo for output Unfinished.

Default disabled

#### 2.7.32 Ex-mode Interface (`CONFIG_EXMODE`)

The ex-mode interface makes a prompt available when pressing *:*. The prompt can be used for entering actions like *:goto-url* and configuration file commands.

The code is still very experimental and lacks much work such as tab completion.

**Default:** disabled

#### 2.7.33 LEDs (`CONFIG_LEDS`)

These are the tiny LED-like indicators, shown at the bottom-right of the screen as [-----]. They are used for indication of various states, e.g. whether you are currently talking through a SSL-secured connection, what is the current input mode (normal or insert), JavaScript errors etc.

**Default:** enabled

#### 2.7.34 Document Marks (`CONFIG_MARKS`)

Makes it possible to set marks in a document and then later jump to them kind of like how fragments in URIs work. It is currently only possible to jump to marks set in the current document.

**Default:** enabled

#### 2.7.35 Debug mode (`CONFIG_DEBUG`)

Assertions are evaluated and will core dump on failure. Some extra sanity checks are done, and some errors will cause core dump instead of just a message. Internal memory leak detection is activated (memory usage will grow), and every allocation/reallocation/free operations will be slower due to extra tests. Lists sanity checks are enabled, so list operations are slower. Hot-key debugging is enabled, it highlights redundant hot-keys in a menu.

This option should be _always_ used by beta testers and developers, it helps to detect many issues. Binary packages maintainers should not use this option in normal situation.

**Default:** disabled

#### 2.7.36 Fast mode (CONFIG_FASTMEM)

This option provides a way to generate a faster and smaller binary of a _stable_ version of ELinks. Please do not use it with unstable releases (unless memory footprint, performance and/or binary size are major issues for you).

It disables all assertion tests and sanity checks effectively reducing safety. It disables internal memory allocation routines, directly calling libc functions (so it's much faster, but memory allocation issues and memory leaks will be not detected). It defines fmem_alloc(), and fmem_free() to be in fact alloca() and nothing, providing much faster allocations in routines where they are used

**Default:** disabled

#### 2.7.37 Own C library functions (CONFIG_OWN_LIBC)

Enable this to use the various C library stub functions that is part of the portability layer instead of those available in the C library on the system.

It will make the binary slightly bigger and should only be used for testing the portability layer.

**Default:** disabled

#### 2.7.38 Small binary (CONFIG_SMALL)

Reduces the size of the binary but also disables a few memory consuming optimizations to make the program much lighter when running.

Part of the size reduction is due to various help text not being compiled in which will affect usability. Also the disabled optimization will make ELinks run slower.

See doc/small.txt for more information about how to reduce the size of ELinks.

**Default:** disabled

#### 2.7.39 Unicode UTF-8 support (CONFIG_UTF8)

By enabling this option you get better Unicode support. At present only some parts of ELinks are influenced with this. It includes DOM, plain, HTML renderer and user interface. Beside normal Unicode characters there is support for double-width characters (like Japanese, etc.).

Some features of Unicode are not handled at all. Combining characters is most visible absence; but see CONFIG_COMBINE below. Some features are partially supported. Like line breaking between double-width characters. There is no other detection for determining when to break or not. Character conversions are still incomplete for ECMAScript strings (bug 805), local file names, and IRIs (RFC 3987).

**Default:** enabled

#### 2.7.40 Unicode combining characters support (CONFIG_COMBINE)

Extends CONFIG_UTF8 with spotty support for combining characters such as U+0303 COMBINING TILDE.

This feature is experimental and has been filed as enhancement 824. Known bugs and weaknesses:

- It assumes wcwidth(wc)==0 means wc is a combining character. However, wcwidth also returns 0 for various control characters (e.g. U+200E LEFT-TO-RIGHT MARK), and apparently returns -1 if LC_CTYPE does not support the wide character. Besides, wchar_t might not be Unicode at all. ELinks should instead use Unicode character properties, perhaps via ICU.
- It assumes all combining characters are nonspacing.
- It works only if the terminal is using the UTF-8 charset.

- It allocates an internal code for each different combining character sequence. A malicious web page could easily use up all the available codes, and the ELinks process would thenceforth be unable to display any new sequences.
- It does not understand canonical equivalences.
- Combining characters work only in HTML text. They do not work in HTML forms, HTML links, HTML document titles, plain text, menus, dialog boxes, or keymaps.

**Default:** disabled

#### 2.7.41 Back-trace Printing (`CONFIG_BACKTRACE`)

Once upon a time, a disaster happens and ELinks crashes. That is a very sad event and it would be very nice to have some means how to diagnose it. In the crash handler, ELinks prints out various helpful things, however the truly important information is _where_ did it crash. Usually, users do not have gdb installed and can't provide a back-trace. However, ELinks can print a back-trace on its own, if the system supports it (currently, it is implemented only for glibc). It is not always accurate, it is useless when the ELinks binary is stripped and it still misses a lot of important information, but it can be sometimes still an indispensable help for the developers.

You should keep this, unless you will strip your ELinks binary anyway, you know you are not going to report back any failures and you care about each single wasted bit.

**Default:** enabled if the libc supports it (only glibc)

#### 2.7.42 Disable Root User (`CONFIG_NO_ROOT_EXEC`)

Browsers are scary monsters used for traveling around in an even more scary world where people indifferently throw garbage files at you and threaten your perfect world. Altho' ELinks is a small monster compared to most browsers, it can still bite your head off and some might consider running it as the root user extremely dangerous. To prevent such usage simply enable this feature.

**Default:** disabled

## 3 Frequently Asked Questions

This is an attempt to capture some of the questions that appear once in a while on the mailing list or on IRC.

### 3.1 I rebuilt/upgraded/reconfigured ELinks and restarted it, but it looks like nothing has changed!

ELinks instances connect together so that they share the cache, bookmarks, cookies, configuration, etc. Only the "master" ELinks instance does any real work and any other ELinkses you run will only connect to this instance. So when you want to restart ELinks, make sure you have no other ELinks instances running.

Alternatively, you can use the `-no-connect` parameter to make ELinks always run standalone, or create a parallel group of ELinks instances with the `-session-ring N` parameter (where `N` is a number larger than zero). Be aware of that in those cases ELinks won't touch any configuration, bookmark, cookies, etc. files. You can force that with the `-touch-files` parameter, but beware that this can result in data loss! (For example, when two master ELinks instances try to save the bookmarks, and you add a bookmark to each of those, only one of the bookmarks survives.)

### 3.2 How does cutting and pasting work?

The reason why normal cut and paste does not work is that ELinks requests all mouse event info even if it is compiled without GPM. Therefore it is necessary to hold down the Shift key when cutting or pasting text. If you do not need to use the mouse for navigation you can disable mouse support by enabling the "Disable mouse" option or passing `--disable-mouse` to configure.

### 3.3 How does the "Move" button on the bookmark manager work?

First you need to mark all the bookmarks (or folders) you want to move. This can be done with the Insert key if you're using the default key-bindings. An asterisk will appear near all marked bookmarks. Now move to where you want to have the stuff moved to, and press the "Move" button.

### 3.4 What's up with the navigation in the various managers?

The managers use list boxes. To navigate (assuming you're using a default key-binding configuration) use Space to open and close the folders.

### 3.5 Why are there so many Links flavors?

Please refer to the history page to read about the various flavors and the differences between them.

### 3.6 Which one to use?

If you want a fast, stable, and feature-thin text browser, use Links-0.99. Main drawbacks include: No HTTP-auth support and no persistent cookies (i.e. they die when all instances of Links are closed).

If you want additional features, including HTTP-auth, persistent cookies, and on-the-fly compression, try ELinks. (Note: links-0.9x and ELinks versions > 0.3 do not share executable name or configuration files so you can safely run these on the same machine.)

If you want the option of graphics mode, use Links2 or Hacked Links. Be aware that some people have reported the latter as hard to configure/compile. You cannot run either of these with Links-0.9x on the same system without hacking the compile-time configuration.

### 3.7 What are the Ki (kibi) and Mi (mebi) units?

They are binary units defined by the International System of Units. Examples:

| Unit | Definition |
|---|---|
| One kibibit | 1 Kibit = 2^10 bit = 1024 bit |
| One mebibyte | 1 MiB = 2^20 B = 1 048 576 B |

### 3.8 How can I get 256 colors?

First, you must enable it in the `feature.conf` file before compiling.

Second, you must run ELinks on a terminal that supports 256 colours:

- Thomas Dickey's XTerm, revision 111. XTerm must be compiled with `--enable-256-color`.
- Recent versions of PuTTY.

Third, you must set the `TERM` environmental variable to `xterm-256color` before running ELinks.

---

**Only 16 colors on The Linux console**
Although the Linux frame-buffer supports 256 (or more) colors, the Linux console driver does not; therefore, console applications are still limited to 16 colors on the Linux console, frame-buffer or not.

---

### 3.9 What User-Agent header does ELinks send?

The older ELinks versions (up to 0.4.3) send:

```
ELinks ($version; $osinfo; $textmode_dimensions)
```

The new ELinks versions (from 0.9.0 on) send:

```
ELinks/$version (textmode; $osinfo; $textmode_dimensions)
```

You should therefore check against something like /^ELinks[\/ ]/, since more fields can be added inside the parenthesis in subsequent versions. Note that users can change their User-Agent through the options system.

### 3.10 ELinks doesn't erase characters from the screen when it should!

When you scroll a web page, you may see ELinks leave some characters on the screen even though it should have erased them. Pressing Ctrl+L usually removes these droppings until you scroll again. There are a few possible reasons:

- ELinks 0.11.* in a UTF-8 locale. By default, ELinks guesses the charset of the terminal from the environment variables LANG, LC_CTYPE, and LC_ALL. ELinks 0.11 versions do not support UTF-8 as this charset. To use ELinks 0.11 on a UTF-8 terminal, you should instead enable UTF-8 I/O via the Setup -> Terminal options dialog box, and choose a charset from Setup -> Character set. This limitation has been removed in ELinks 0.12pre1.
- Web pages may use nonspacing combining characters or Unicode control characters that ELinks does not recognize as such. This happens especially on http://en.wikipedia.org/wiki/Special:RecentChanges, where the server generates U+200E LEFT-TO-RIGHT MARK characters. ELinks 0.13.GIT now has some support for these characters; see ELinks bug 824.
- Some versions of the Terminal application in Mac OS X appear to have a setting that makes line-drawing characters take up the space of two ASCII letters. ELinks does not expect this. To avoid the incompatibility, either disable the setting in the Terminal application or select "No frames" in the Terminal options dialog box of ELinks.

## 4 Introduction to the World of ELinks

The goal of this introduction is to explain the basic concepts in ELinks, give an overview of how to get started and serve as an entry point to many of the (undocumented) features of ELinks. It won't tell you all the details, but should hopefully give you an idea of how things work and make it possible for you to even figure out how to go further.

Although ELinks is text-based, the user interface has many of interaction methods normally found in graphical environments. There are menus, dialogs with buttons and hierarchic list boxes with folders. Care has been taken to make the interaction between various dialogs consistent, so the controls will quickly become familiar to new users.

The user interface can be controlled using both mouse and keyboard, but currently it is only possible to configure keybindings. Looking back, the key-controls have been more advanced than the mouse support, but during the 0.10 prereleases the mouse support has been much improved. You will now find find stuff like contextual menus when right-clicking in different document zones.

### 4.1 Overview of the User Interface

The main user interface of ELinks consists of the document view and dialog bars displaying the information such as the title of the currently viewed document, all opened tabs and the browsing status. The 3 bars are elaborated further below.

The most important dialogs that you will meet include the Main, Link and Tab menus and the different managers. The menus serve as entry points to the actions available from different contexts, while the managers let you check the state and control the various subsystems, such as loaded cookies and the global history. The utility menus and the manager tools are investigated further below.

The document viewer in ELinks provides a feature-rich set of ways to browse documents. That is, multiple options exist for navigating, searching and displaying documents and you will hopefully figure in time what works best for you. The basic browsing possibilities are presented below.

ELinks is highly configurable, so if there is something that you would like to change, it is most likely possible. The best overview of the many options are given in the `elinks.conf(5)` man page. Keybindings are discussed in the `elinkskeys(5)` man page. It is not always up-to-date, so you should also check the keybinding manager and the overview of all the configured keybindings given in the Help -> Keys dialog. The Keys dialogs serves as a good introduction to the most common keybindings.

## 4.2 The Title, Tab and Status bar

The title bars main purpose is to display the title of the current document. Documents bigger than can be displayed with the current screen size are divided into subpages. In this case the current document position is indicated in the far right of the title bar as a suffix to the actual document title. The syntax is: ( current-subpage / total-subpages ), an example is `(4/9)` that indicates the 4th subpage of 9 subpages.

The tab bar by default is only visible when 2 or more tabs are open. It is divided into slots containing the trimmed title of the tabs' loaded document. Between each tab is a separator. The current tab is highlighted and all tabs that has not been viewed after being loaded are highlighted as fresh. Tabs are explained in details in the tabs.txt file.

The status bar has multiple purposes. Most of the time it will contain the URI (and title) of the currently selected link. If a link is followed, connection information is shown in the status bar. When using cursor routing, the status bar will show the coordinates of the cursor when a link is not followed.

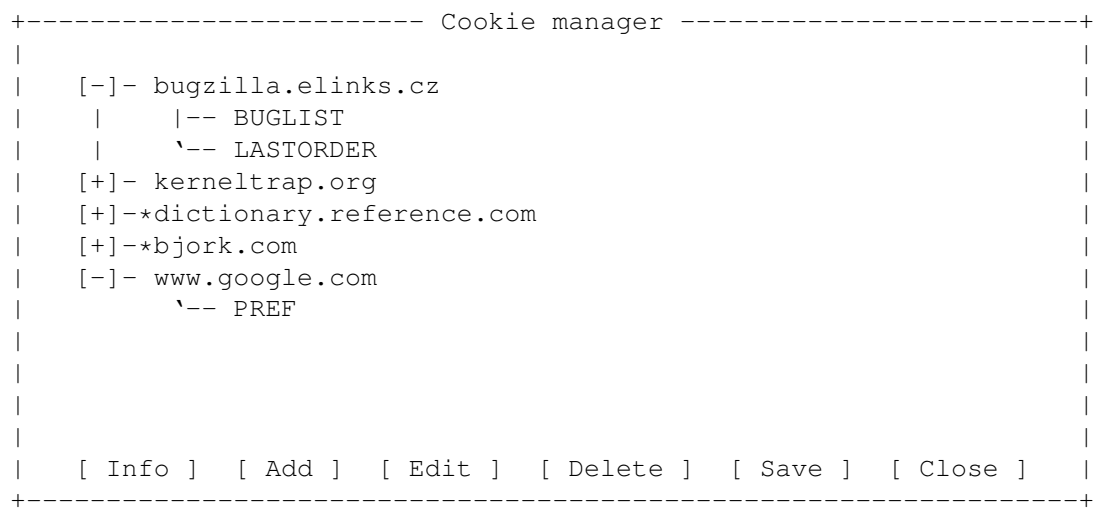
## 4.3 The Main, Link and Tab Menus

The Main Menu gives you access to many of the features of ELinks via submenus and serves as a good entry point for performing actions on the different objects of interest, such as links, documents and tabs. The Link menu and Tab menus are more specialized and targeted to a specific context, such as link handling or managing the tab bar. Actually, the Link Menu is accessible from the Main Menu as a submenu.

Once you have familiarized yourself with the menus, you will have a good chance at gradually learning the most common keybinding, since all the configured keybindings are shown as right aligned text. Menu entries can also be quickly accessed using hotkeys. Hotkeys appear highlighted in the menu entry text. For example the key-combo "Alt-v i" will open the document info dialog accessible from the View sub menu in the Main menu.

## 4.4 The Managers

The managers let you control the state of subsystems, such as cookies and the global history. They are accessible from the Tools or Setup submenu in the Main Menu. The managers consists of an area showing a hierarchic listbox and buttons at the bottom. Below, a view of the cookie manager is shown.



Each item is either a folder or a leaf. A folder is displayed with a `[-]` or `[+]` before the name telling whether the folder is currently open or closed. Nested items are displayed indented compared to the folder they are nested in. In the cookie manager example above "bjork.com" is a folder and "PREF" is a leaf.

Items can be "marked", which makes it possible to select a group of items and perform an action on them, such as deleting all marked items. If any item has been marked the currently selected item is ignored when performing the action. Marked items are displayed with an asterisk (*) prefixing the name.

The buttons make it possible to perform actions either on selected or marked items or on all items in the manager. Buttons named *Clear* and *Save* are performed on all items; *Clear* will delete all items and *Save* will update the runtime state file associated with the manager in the `~/.config/elinks/` directory. Most buttons presses will query you before completing the action.

At any time, both the currently selected item and button are highlighted. The same goes for marked items. Most manager dialogs also maintains the state, so that when you reopen the manager later it will have the same items selected and the same folders opened or closed.

The basic default controls for managers are the following:

| Keys | Action |
|---|---|
| Up/Down | Select the item above/below. |
| * | Toggle marking of a item. |
| Space | Open and close folders. |
| Left/Right | Select the button to the left/right. |
| Home/End | Select the first/last item. |
| Enter | Press the currently selected button. |
| Esc | Close the manager dialog. |

Some managers also supports searching, either by pressing the *Search* button or by pressing */*. By searching the empty string, all hidden items from the previous search will be shown again.

## 4.5 LED status indicators

As an optional feature it is possible to have tiny LED-like status indicators shown at the bottom-right of the screen. They are used for displaying an overview of the current browsing state, ie. whether you are currently talking through a SSL-secured connection, what is the current input mode (normal or insert), JavaScript errors etc.

An example display may look like: `[SIJP--]`. Each position in the LED display is associated with the following state:

| Symbol | Description |
|---|---|
| *S* | Whether an SSL connection was used. |
| *i/I* | The state of insert mode for text-input form-fields: *i* means modeless, *I* means insert mode is on. |
| *J* | A JavaScript error has occurred. |
| *P* | A JavaScript pop-up window was blocked. |
| - | Unused. |
| - | Unused. |

- generally indicates that the LED is off.

The above information is also available in the LED dialog available by either clicking on the LED display or via the Help menu.

## 4.6 Navigation

ELinks provides various ways to navigate documents. Depending on how documents are structured, it can be a great help to change navigation style. The navigation styles can roughly be divided into page-oriented, link-oriented and screen-oriented. They overlap in many ways, so this separation is mostly used as a mean to present them.

#### 4.6.1 Page-Oriented Navigation

This involves scrolling documents horizontally and vertically. Documents can be scrolled page-wise, where the next or previous subpage will be displayed. It is also possible to scroll documents in steps, either line-wise (vertically) or column-wise (horizontally). The step size can be configured and by default is 2 lines and 8 columns. Alternatively, whole documents can be scrolled to the start or the end.

The basic default controls:

| Keys | Action |
|---|---|
| Insert/Delete | Scroll up/down line-wise. (vertically) |
| PageUp/PageDown | Scroll up/down page-wise. |
| *[/]* | Scroll left/right column-wise. (horizontally) |
| Home/End | Scroll to the start/end of the document. |

#### 4.6.2 Link-Oriented Navigation

For hypertext documents, access to the links makes it more practical to navigate by jumping between links in the document. There are two ways to do this; either you can move between links relationally or by number. Using relational link navigation it is possible to focus the next/previous link or move in a directional manner to the link in a certain direction such as left/right/up/down.

In order to navigate using link numbers, you have to first toggle link numbering on; this will prefix all links with a number using the notation [number]. `[23]` indicates link number 23. When link numbering is enabled, pressing any number key will pop up a "Go to link"-dialog where the complete link number can be entered. By pressing Enter the entered link number will be focused, but only if it is a valid link number.

Note: it is also possible to jump to links by searching the link text; check the documentation on searching.

The basic default controls:

| Keys | Action |
|---|---|
| Up/Down | Move to the previous/next link. |
| . | Toggle link numbering. |
| Enter/Right | Follow the current focused link. |

No keys are by default configured for directional link navigation.

#### 4.6.3 Position-Oriented Navigation

Positional navigation (sorry, bad word) uses the either the position of the cursor or the mouse click to navigate the document. If you are familiar with the w3m text-browser you will be familiar with cursor routing. Basically, you move the cursor around — kind of like a mouse — in the document area of the user interface. When the cursor is over a link, the link is highlighted, and when the cursor moves outside the current document view, it will cause the document view to scroll.

The possibilities when using the mouse to navigate the document depend on what terminal you are using. In some terminals, it is possible to scroll by using the mouse wheel. Scrolling is however also possible by clicking in the edge areas of the document view. Highlighting links can be done by clicking on a link but waiting to release the mouse button until the link is no longer under the mouse pointer.

No keys are by default configured for cursor routing.

#### 4.6.4 Forms

The status bar will indicate the type and name of the field.

**Input text/Password fields** These will be displayed as _____. Note that passwords will be obscured using * characters. Status bar will display something like "Text field, name q", or "Password field, name password" for password fields.

**Textarea boxes** These will be displayed as multiple lines consisting of _. Status bar will display something like "Text area, name comment"

**Buttons** These will be displayed as `[ Go ]`. Status bar will display something like "Submit form to ...", "Post form to ..." for submit buttons.

**Checkboxes** These will be displayed as `[ ]` or `[X]`. Status bar will display something like "Checkbox, name c, value 1". To set one just press ENTER on it.

**Radio buttons** These will be displayed as `( )` or `(X)`. Status bar will display something like "Radio button, name radio1". To set one, you may use ENTER.

**Select lists** These will be displayed as `[first item____]`. Note that if multiple attribute is used, these are displayed as a group of checkboxes instead. Status bar will display something like "Select field, name list" To select one entry, press ENTER, then navigate using UP/DOWN, then press ENTER again.

## 4.7 Searching

Searching is by default available by pressing */*. This will open a search dialog with a input text field for entering the search terms and checkboxes to control how searching is performed. You can indicate whether matching should be case sensitive and whether regular expressions or normal searching should be used.

It is also possible to make an incremental search, also called type-ahead searching. You can search either the whole document text or only link text. The latter can be useful if you see a link deep inside a page and want to get to it quickly.

Matches of the search term will be high-lighted. After having performed document text search all matches will be high-lighted. To get rid of this high-lighting you have to “search for the empty string”, that is open a search dialog and just press Enter in the input field.

Previous search words are saved in the search history, so they can easily be found and used later. Browsing the history will replace the current entered search terms.

The basic default controls for searching are the following:

| Keys | Action |
|---|---|
| */* | Open search dialog |
| *?* | Open search dialog for backwards searching |
| *#* | Start incremental link text search |
| *#/* | Start incremental document search |
| *n*/*N* | Show next/previous match |
| Tab | Show next match (only for incremental searching) |
| Up/Down | Insert previous/next search word from history (only when the input field is selected) |

## 4.8 Hints and Odd Features

Note: This is still a work in progress and from here on an below everything is marked TODO!

- Numerical action prefixes. Example: 3<Down> jumps down three links.
- How to move forward in the document history (*u*).
- Toggling color modes, plain/html and image link rendering.
- Link numbering.
- Insert mode in text-input form-fields.
- Menu searching.

# 5 The Ultimate Bookmarks Guide

Glad to see you again, mortal. Now, we are going to learn about bookmarks - how to use them, how to maintain them, and also something more about the file formats and maybe even about the code structure later. But don't fear, we won't burden you with it, just stop reading when you already know enough.

In order to read this, you need some common sense, the ability to start ELinks and some idea about what's a Web document, a URL address and knowledge like that.

If we ever mention some keys here, please note that you can rebind almost any action to another key which you like more - then you must obviously imagine your own key in place of that. Below, we will list all actions, options and so on related to bookmarks. We won't tell you how to rebind the keys, though; another document will take on that.

Somewhat out-of-order, a very frequent FAQ: In order to move bookmarks around, you need to mark them first - press *Insert* or * (if you use the default keymap) to do that.

## 5.1 The Bookmark Manager

Basically, almost everything is going on in the so-called bookmark manager. That's a special dialog window, which contains a listing of all the bookmarks you ever told ELinks to remember and it lets you to do with them anything you would ever want to do with them.

You launch the bookmark manager by pressing the *s* key in standby (standard) mode. You should see a big empty space (bookmarks will slowly appear there as you will add them) and couple of buttons shriveling at the bottom. So, as a start, move with the right (or left; both will do) arrow to the button **Add bookmark** and fill in the input fields it offers to you. I mean, you can type something like "ELinks homepage" to the first field, then move down by e.g. the down arrow and fill "http://elinks.cz/" to the second field. Then, bravely press enter and watch the bookmark popping up at the top of the vast area reserved for bookmarks.

Repeat this step few times. Now, you can move between bookmarks by the up and down arrow, jump to the location any of them points to by the Goto button, change it by the Edit button, delete it with the Delete button and so on. When you'll become bored, press the escape button and you're free again!

## 5.2 The Ancient Forests

It's not very convenient to have all the bookmarks mixed up - soon, you will get lost in them. Thus, in ELinks you can categorize them to various folders, subfolders, subsubfolders and so on, then you can expand and back enfold them and so on.

In order to create your first folder, use the button **Add folder** and fill the first input field. You can safely ignore the URL field, ELinks will do the same. **POOF** and you see it - it has that strange `[+]` or `[-]` thing there. If it has `[+]` near, it's enfolded, while when it has `[-]` near, it is expanded, while you can change that by pressing the spacebar.

In order to add a bookmark into a folder, move on the item of the folder (it must be expanded) or onto any bookmark inside of the folder and simply do the usual **Add bookmark** job. You can also move the bookmarks around, obviously. Before pressing the **Move** button, you need to first mark all the bookmarks (or even folders) you want to move using the *Insert* or * key—asterisk will appear near of all marked bookmarks—and then move to where you want to have the stuff moved to.

Separators can be inserted as well, using **Add separator** button, or by entering a special bookmark with "-" as title and no url.

## 5.3 Searching for a needle in the haystack

Of course, you can search in the bookmarks. Just use the **Find** button - for convenience, you have the current document's URL and title pre-filled there, and for convenience only up-up-enter-down-down sequence is enough to have the playground clean. Then, just fill a substring of what you are looking for, and the bookmarks will be filtered so that only the matching ones are shown. (Actually, currently it will not be filtered but the cursor will only jump to the first matching bookmark below the current cursor position - and it will **NOT** wrap around. The exact behaviour changes time by time and hasn't been stabilized yet.)

## 5.4 File formats

ELinks supports two bookmark formats: the native format and a generic bookmark exchange format called XBEL. Each of those formats has its pros and cons, which we shall discuss below. You can switch between then by changing the option *bookmarks.file_format*.

However, first please note that ELinks *CANNOT* read Links bookmarks directly. Importing Links-0.9x (or Links-1.x) bookmarks is easy - it is just matter of changing all the | (pipe) characters to tabs. There is a script for that in the contrib/conv/ directory. Importing Links-2.xx bookmarks is not so easy; in fact, the scribe knows of no way of doing that at the time of writing this, so this is up to you to figure out (if you do, please tell us so that we can add it here). Perhaps you might find a way to convert Links2 bookmarks to the XBEL format, which can then be loaded in ELinks.

### 5.4.1 Native file format

This is the preferred bookmarks format, which is also used by default. The bookmarks file is `~/.config/elinks/bookmarks`, in a simple format:

```
<name> '\t' <url> [ '\t' <depth> ['\t' <flags>] ] '\n'
```

\t represents a tab character, \n represents a newline character. [Square brackets] denote optional parts. The <*name*> and <*url*> fields should be obvious. <*depth*> contains the depth level of the entry - by that, ELinks can unambiguously figure out the bookmarks hierarchy:

```
Bookmarks structure:                              Depth:
 ,-- The Pasky's C Bestiary                       0
[-]- Wonderful things                             0
 |    |-- Christmas Carol in l33tsp34k by L.M.     1
 |   [-]- Beautiful Potato Camera Shots           1
 |   [-]- Gallery of Scary Images of Jonas Fonseca 1
 |         |-- Jonas torturing gdb                2
 |        [-]- Urgh                               2
 |         |    `-- Jonas consuming Tofu          3
 |         `-- Jonas with crashed ELinks          2
 |-- Slides from Witek's hack-patch show          0
 `-- Miciah's English Grammar Spellbook           0
```

<*flags*> is a string of characters. Currently, two flags are supported:

| Flag | Description |
|---|---|
| E | This folder is currently expanded. (No effect for non-folders.) |
| F | This entry is a folder. The <url> part is usually empty. |

Separators: these are special bookmarks with "-" as title and no url.

**Pros** Naturally, ELinks handles the native format the best, easiest and most reliably.

**Cons** It is unlikely that you could use the native format anywhere else than in ELinks.

To use the native format, set *bookmarks.file_format* = 0.

### 5.4.2 XBEL file format

The XBEL file format support was added at some point during the 0.4 development by Fabio Boneli. It has never been complete and has plenty of troubles, but generally, it works at the basic level. The bookmarks file is `~/.config/elinks/bookmarks.xbel` (thanks to a different filename, you can have both XBEL and native bookmarks saved in your `~/.config/elinks` directory).

We shall not describe the XBEL file format here,

`http://pyxml.sourceforge.net/topics/xbel/`

is the authoritative resource on that. It also contains list of some of the applications supporting the format. Basically, you will be probably able to convert from/to the XBEL format to/from most of the other widely used formats, so this way you can import your bookmarks to ELinks from basically anything.

**Pros** XBEL is the gateway to the rest of the bookmarks world.

**Cons** The support for XBEL is incomplete and there are known bugs. Especially, national character sets are basically not supported, so ELinks will most likely get it wrong if you have any non-ASCII characters in your bookmarks. Generally, the XBEL support should be considered experimental and you shouldn't rely on it. It **could** trash your XBEL bookmarks file so make regular backups.

To use the XBEL format, set *bookmarks.file_format* to 1.

#### 5.4.3 Usage hints

As already noted above, probably the best usage pattern is to use XBEL for importing/exporting your bookmarks to/from ELinks and the native format for regular operation. Of course, if you want to synchronize your bookmarks in ELinks and some other XBEL-supporting gadget and you are brave, you can use XBEL as your exclusive bookmark format - the choice is upon you.

Regarding the bookmarks synchronization, there is one important note. ELinks saves your bookmarks each time you added one through the *a* shortcut (add-bookmark action) or when closing the bookmarks manager if you made any changes or when quitting ELinks. However, ELinks reads your bookmarks only **ONCE**, during the startup. This behaviour may change in the future (tell us if you need a way for ELinks to re-read the bookmarks file), but this is how it is done now.

Actually, you may now ask "So how do I convert bookmarks between the two formats?". It is quite easy. ELinks simply follows the current value of *bookmarks.file_format* whenever loading/saving the bookmarks.

So, e.g. if you normally use the native format but you want the bookmarks to be saved in the XBEL format once, change *bookmarks.file_format* to 1, then cause the bookmarks to be resaved (e.g. by doing some simple change, like adding a trailing space to some bookmark's title or so), then change the *bookmarks.file_format* value back to 0.

It is a little more complicated if you normally use the native format but you want to import bookmarks from the XBEL format once. You again change *bookmarks.file_format* to 1, then cause the bookmarks to be reloaded. That involves saving the configuration, quitting ELinks _completely_ (that means closing/killing all instances of it you have running), then restarting it and changing *bookmarks.file_format* to 0. Then save the configuration again and cause ELinks to resave the bookmarks.

Agreed, this all strange dances are quite clumsy, likely some simple wizard-like interface for switching the bookmarks formats will be introduced in the future. So far, we have had no reports from our users that anyone wants to switch their bookmarks format frequently, so this is not too high on our TODO list. So be sure to tell us if you would like this process to be simplified rather sooner than later.

## 6 The Wonders of Tabbed Browsing

In this information age with our stream of consciousness constantly being dispersed by links to different resources on the Net, it is a challenge to keep track of where you are going. The need for being able to access several pages in parallel arises. Tabbed browsing gives you an easy way to browse multiple sites in parallel.

If you are not already familiar with the concept of tabbed browsing you can think of a tab as a separate browsing context with its own history and various other browsing state information, such as search word and document loading. Whenever you stumble upon a link to a document that you want to follow without leaving the current document, you can open it in a new tab. This makes it possible to more easily jump between pages on the Net and removes the need for running more than one ELinks for that purpose.

Options related to tabs are located under "User Interface -> Window Tabs" in the option manager. In the configuration file the naming prefix is "ui.tabs".

Tabbed browsing has been supported since version 0.9.0 and is fairly complete. The documentation on tabs is therefore divided into two chapters: a general introduction and an introduction to advanced topics.

## 6.1 Introduction to the basic actions involving tabs

### 6.1.1 The tab bar and the tab menu

The current state of all opened tabs are displayed in the tab bar. The tab bar will, by default, become visible when more than one tab is open, but this is configurable. For each open tab, the document title will be shown, possibly truncated. The current tab is highlighted. The tab bar will also display a load meter for tabs that are loading documents. Finally, any tab that has not been selected since its document was loaded will be marked as "fresh" by using a different coloring scheme.

The tab menu gives access to tab specific actions along with some other useful document specific actions. So even if you haven't configured keybindings for all actions, chances are you will find it in the tab menu. By default, it is opened by pressing *e*.

### 6.1.2 Creating new tabs

When creating new tabs, it is possible to specify whether to create the tab and make it the current active tab or if the tab is to be created "in the background" — that is, without it taking over the focus.

Tabs can be created either with or without specifying a desired first document to load. That is, you can open links or submitted forms in a new tab or just open a new tab. Depending on your configuration, the latter will load the configured homepage in the newly created tab or simply leave the tab blank with no loaded document.

By default, *t* will open a new tab and *T* will open the current link in a new backgrounded tab. You can configure keybindings for opening a new tab in the background and opening the current link as the active tab.

### 6.1.3 Switching between tabs

By default, it is possible to switch between tabs by using < and > to select the previous and next tab, respectively. When positioned at the leftmost tab, and switching to the previous tab, the tab switching will perform a wrap-around so that the rightmost tab will be selected. The wrap-around behaviour is configurable.

### 6.1.4 Closing tabs

Tabs can by default be closed by pressing *c*. It is possible to optionally have a confirmation dialog pop up when closing a tab to avoid accidental closing. To complement closing of the current tab, it is also possible to close all tabs but the current one. No key is by default configured for this; the tab menu, however, provides this ability.

Note: downloads initiated from a tab are in no way tied to that tab, so tabs can be closed and the download will not be affected.

## 6.2 Advanced topics involving tabs

### 6.2.1 Moving tabs

Newly created tabs are always positioned as the rightmost tab, but it is possible to move the current tab either to the left or the right. The default keybindings have them bound to Alt-< and Alt->. Note, however, that there are problems recognizing those keybindings when using XTerm, so you might want to rebind them.

### 6.2.2 Saving and restoring tabs

Several features use bookmarks to save tabs; they will create a folder and bookmark therein the currently displayed document of each tab:

- You can explicitly command all tabs to be bookmarked. This will ask you for a folder name in which the tabs will be bookmarked.

- At startup and shutdown tabs can automatically be bookmarked in order to save and restore the browsing state. Note that when restoring, all history information will be gone. It is possible to configure both tab saving and restoring via options in “UI -> Sessions”.
- As a mean of crash protection, tabs can periodically be saved so that it is later possible to reconstruct opened tabs. In case of a clean shutdown periodically saved tabs will be removed.

# 7 Marks (the lite edition)

So, were you ever reading this huge 300-pages specification heavily cross-referencing itself, jumping around and getting a headache when looking for the place where you stopped reading the last time?

Were you doing something similar in C code, but praising **vi** for document marks?

ELinks can do them, too! For vim non-users:

## 7.1 What it is?

When you place a "document mark" (just "mark" from now on), you place an _invisible_ anchor at the current position in the document. You can place several such marks --- each mark is identified by a single character (any reasonable character will do). Then, you can just happily browse around aimlessly (but see below) and when in the same document again, you can return to any of the marks in the file again. That will restore your position in the file at the time of placing the mark.

You can place a mark by the "m" key followed by the mark character. You can go to a mark by the "'" (apostrophe) key followed by the mark character. E.g., you can place a mark named "a" in the file by pressing "ma", then return to it anytime later by typing "'a". You can of course change those shortcuts at any time to anything you wish in the keybindings manager.

Short summary: you can place a mark (e.g. *z*) in a document by pressing "mz" and then go back to it anytime later by pressing "'z".

## 7.2 Restrictions

Currently, only A-Za-z characters are valid marks.

Only one mark named "a" (or anything else) may exist at a time, so if one puts a mark "a" in a first document and set another mark "a" in a second document, ELinks will simply replace the former one.

ALL the document marks are always local to the document. I.e. the vim text editor has an extension that makes the capital-letter marks to be global to the whole program and going to such a mark will make it to open the right document. This is not implemented in ELinks _yet_.

Contrary to vim, ELinks doesn't support numbered marks (jumping to the last n documents in history) nor the special "'" mark (jumping to the last mark). Yet.

There is no way to get a listing of all marks set in a document. Yet.

## 7.3 Marks lifespan

I already hinted something about another restriction regarding aimless browsing. The lifespan of document marks depends on rather ill-defined and (for an average mortal) mostly non-deterministic technical conditions.

Generally, marks _always_ survive when not moving away from the document or when moving only in the session history (and unhistory). That means, if you go back and the "unback" to the document, you will find your marks safely in place. If you follow a link from the document (or typed an address to the Goto URL dialog) and then go back (by pressing the right arrow or through the File menu), your marks are safe too. These are in fact by far the most common usage cases for the marks, so most of the time it will just work as you expect. That's a good news.

The bad news is that in all other cases, nothing is guaranteed. It might work if you get back to the document by any other means (by following some link or typing its address to the Goto URL dialog), or it might not. It might be possible to achieve two

instances of the document inside a single ELinks, each with its own set of marks. However, again, generally it will work as expected - this paragraph serves only as a disclaimer in cases it doesn't. Don't rely on it.

Marks never survive over ELinks restarts. If you quit your ELinks completely and run it again, the marks you placed will be no more. No exceptions. Well. In some cases, it **might** appear that they survived, but that just means you did not quit your ELinks _completely_ --- if you run multiple ELinks instances under a single user on a single system, they "join" together and you must quit (or kill) them all to get rid of the damn thing. But that's a different story.

# 8 URL Shortcuts in ELinks

One extremely useful and convenient feature in ELinks, which may not be very well known, is so-called URL rewriting. If you give ELinks a URL (by passing it on the command line, through the -remote magic device or, most frequently, by typing it to the Goto URL dialog), it has the ability to mangle it in certain ways before processing it. This mangling is called URI rewriting.

## 8.1 What it does

The URI rewriting can expand certain URL shortcuts to complete URLs. For example, if you type *sd* to the Goto URL dialog, it will be rewritten to:

```
http://www.slashdot.org/
```

first, and then loaded. /.'s front page will be displayed.

Further, if you type *g elinks* to the Goto URL dialog, it will be rewritten to:

```
http://www.google.com/search?q=elinks&amp;btnG=Google+Search
```

and then loaded, therefore, a Google search for the keyword *elinks* will be performed. Note that you can separate the *g* by either a space or a colon, so *g:elinks* will do the exact same thing.

A large number of those shortcuts are already defined for you. You can browse Protocols :: URI Rewriting :: Dumb/Smart prefixes in the Options Manager for a complete listing of already defined shortcuts (press the [Info] button on a shortcut to see what it will be rewritten to). See below for details on how to enable or disable this rewriting and how to define your own shortcuts.

## 8.2 How it works

All the URI rewriting can be controlled by options in the *protocol.rewrite* option tree.

In the case of *sd*, a URI rewriting mechanism called *dumb prefixes* is used. If protocol.rewrite.enable-dumb is true, ELinks looks at the contents of the protocol.rewrite.dumb.* tree, and if it finds option protocol.rewrite.dumb.sd, it will use its value as the target URI.

Therefore, more generally speaking, if ELinks looks at protocol.rewrite.dumb.<typed_string>, and if it exists, will replace the entered URI with it. These dumb prefixes can be used as a kind of alternative bookmark systen - if you want to have fast access to a certain site and you don't want to spend a while navigating the bookmarks manager, you can just fire up the Goto URL dialog, type the shortcut and there you go.

A dumb prefix can contain *%c*, which will be replaced by the URI of the current document. This is used, for example, in the *arc* dumb-prefix, which provides a shortcut to the Wayback machine at archive.org.

In the case of *g*, a slightly different URI rewriting mechanism called *smart prefixes* is used. If protocol.rewrite.enable-smart is true, ELinks looks at the contents of the protocol.rewrite.smart tree and, if it finds option protocol.rewrite.smart.g, it will use its value as the target URI.

The difference is that the smart prefixes can take arguments and reference them in the target URI. As with dumb prefixes, *%c* in the URI means the current address. Additionally, *%s* will be substituted by the whole string following the prefix (*elinks* in the *g* example above; *%s* is by far the most frequently used expansion), while you can reference individual space-separated arguments with the codes *%0* thru *%9*. Finally, if you want a plain % to the resulting URI, use *%%*.

Note that the prefix searched in the protocol.rewrite.smart tree is everything in the typed address up to the first space or colon (so *bug:123* and *bug 123* are identical). These prefixes are generally useful for searching anything very fast - be it Google (*g:foo*, *gi:foo*, *gr:foo*, ...), ELinks Bugzilla bugs (*bug:1234*), the RFC database (*cr:foo*), or the Czech-English dictionary (*czen:foo*). The only limit is your imagination.

### 8.3 Alternative URI rewriting mechanisms

In the past, before this was implemented, Lua scripting did the job. And you still have complete control over the URI in the internal scripting goto-url hook. The advantages are clear - you get complete control over the URI and you can do many more interesting things with it. For example, there are some very advanced CVSweb and Debian package database URI prefixes implemented in the sample Lua hooks file. The one disadvantage to this is that you must have Lua scripting enabled in order to make use of it, and many users don't have Lua installed.

## 9 The Terminal Setup

ELinks uses neither (n)curses nor termcap/terminfo, so unless you are using a terminal that is known by ELinks and have a built-in configuration (see below for a list), it might be required that you do a little configuring of how your terminal should be handled. The easiest way to do this is by using the Terminal Options dialog located in the Setup submenu of the Main menu.

It is possible to have configurations for multiple terminals as long as the TERM environment variable — used to distinguish terminals from one another — is set to something different for each terminal. So be sure to set TERM to different values for two terminals that cannot share the same configuration. For example, always setting TERM to xterm-color can lead to problems if you run ELinks under the Linux console. This is because the Linux console does not support underlining and ELinks will not know that underlined characters will have to be color highlighted so they stand out.

In short, ELinks does not use termcap or terminfo. ELinks uses $TERM, but only to distinguish between terminals in its own configuration. That is, you need only configure your terminal within ELinks: Go to the Setup menu and select Terminal Options. If $TERM is set to *screen* when you configure ELinks' terminal settings, ELinks will remember to use those settings when $TERM is *screen*.

### 9.1 Options

Apart from the last charset option configurable through the Setup -> Character Set submenu in the Main menu, the rest can be configured using the Terminal Options dialog.

#### 9.1.1 Terminal type

It maps roughly to the terminal type, such as Linux console, XTerm, VT100 etc. It matters mostly when drawing frames and borders around dialog windows. As already mentioned, it also turns on certain features which try to compensate for "missing" terminal capabilities when drawing. Special highlighting of underlined text if underlining is not supported is one such thing.

| Type | Notes |
|---|---|
| No frames | Dumb terminal type / ASCII art |
| VT 100 frames | Works in most terminals |
| Linux or OS/2 frames | Linux console / you get double frames and other goodies |
| KOI-8 | |
| FreeBSD | FreeBSD console |

The default is to assume dumb terminal/ASCII art.

#### 9.1.2 Color mode

The color mode controls what colors are used and how they are output to the terminal. The available color modes are:

| Mode | Color codes |
|---|---|
| Mono mode | Only 2 colors are used |
| 16 color mode | Uses the common ANSI colors |
| 88 color mode | Uses XTerm RGB codes (if compiled in) |
| 256 color mode | Uses XTerm RGB codes (if compiled in) |

The default is to use mono mode.

#### 9.1.3 Switch fonts for line drawing (aka 11m hack)

This is related to the drawing of frames and window borders controlled by the option above. It controls whether to switch fonts when drawing lines, enabling both local characters and lines working at the same time.

This boolean option only makes sense with the Linux console. Off by default.

#### 9.1.4 Restrict frames in cp850/852

This is related to the drawing of frames and window borders controlled by the option above. If enabled, it restricts the characters used when drawing lines.

This boolean option only makes sense with the Linux console using the cp850/852 character sets. Off by default.

#### 9.1.5 Block cursor

Move cursor to the bottom right corner when done drawing, if possible. This is particularly useful when we have a block cursor, so that text colors are displayed correctly. If you are using a screen reader you do not want to enable this since the cursor is strategically positioned near relevant text of selected dialog elements.

This boolean option is off by default.

#### 9.1.6 Transparency

Determines whether or not to set the background to black. This is particularly useful when using a terminal (typically in some windowing environment) with a background image or a transparent background. If this option is enabled the background will be visible in ELinks as well.

Note that this boolean option makes sense only when colors are enabled. On by default.

#### 9.1.7 Text underlining capability

This boolean option controls whether to underline text or instead, enhance the color.

Note: not all terminals support text underlining, so it is off by default.

#### 9.1.8 UTF-8 I/O

This boolean option controls outputting of I/O in UTF-8 for Unicode terminals.

Note that currently, only a subset of UTF-8 according to the terminal codepage is used. This is off by default.

#### 9.1.9 Character Set

Codepage of the charset used for displaying content on terminal.

The value *System* (which is the default) will set the charset according to the current locale. The default is to use the charset of the current locale.

### 9.2 Terminal Configurations

Built-in configurations exists for the following terminals:

- linux
- vt100
- vt110
- xterm
- xterm-color
- xterm-88color
- xterm-256color

The last two configurations requires that support for either 88 or 256 colors is compiled in to xterm.

GNU Screen is VT100-compatible, so select *VT 100 frames*. GNU Screen also supports colors just fine, so select *16 colors*, or, if you are running Screen within a terminal emulator that supports 256 colors and you have compiled both Screen and ELinks to support it, choose *256 colors*.

## 10 Introduction to MIME handling

At some time along the 0.4 prereleases, ELinks replaced the old Links file configuration system with its own one. Now, the configuration is stored in elinks.conf, much more complete and featuring a new syntax. You can set most options from the UI (so usually there is no need to deal with elinks.conf), and that used to be true for setting MIME handlers and extensions as well; but now the associations menu is gone temporarily, forcing you to set the handlers from the configuration file.

The comments provided within elinks.conf for the MIME options are not much helpful, and those options may seem a bit obscure or confusing at first. This document is a basic introduction to setting of MIME associations for ELinks. If you want to know how to automatically launch a program to view some kind of file (like images), then read on. Some basic knowledge of MIME is recommended.

### 10.1 Handling MIME types, the ELinks way

#### 10.1.1 What are MIME types and why may you want to use them?

If, while browsing with ELinks, you need to display files that it can't show (like images or PDF documents), then you will likely want to pass them to other programs that are suitable for this task. While you may do it manually (saving the file to disk, running the auxiliary program to show it and then removing the file when finished), ELinks provides a method to do this automatically using MIME types.

ELinks usually knows what the MIME type is for any file (which is a kind of description of its format), thus you only need to specify how to manage the MIME types for the files you want to open. If you don't tell ELinks how to manage a given MIME type then it will show you on screen the contents of the file (as if it was ASCII text) instead of calling an external program to show them.

#### 10.1.2 Associating files to MIME types

If the file is fetched from a web server then this web server should tell ELinks what its MIME type is, so you should have generally no need to care about it. However, the web server might send an incorrect type, or you may be retrieving it by FTP or from your local filesystem, so sometimes ELinks needs to guess it.

The easiest method to guess a MIME type for a file is to just look at its name trusting it was given the right extension. ELinks supports this, letting you to specify a type for any given extension.

#### 10.1.3 Managing a given MIME type

This usually means specifying programs to open files of this type and the conditions in which those programs may be used. For instance, you may want to use zgv (a popular svgalib image viewer) when you are using the text console, but xli (a image viewer for X) when running the X window system.

##### 10.1.3.1 Binding it to a handler

Instead of directly attaching a program to a MIME type, ELinks lets you associate an abstract handler to each type. Then you can specify which programs implement the handler and how are they used.

For instance, you may create an image_viewer handler and assign it to MIME types image/jpeg, image/png and image/gif. Then you would associate to image_viewer the programs you would like to use for viewing images and the details on how to use them. This is less cumbersome than repeating all these details for each MIME type.

#### 10.1.4 Specifying the details for a handler

There are three issues to specify for a handler: the program associated to it, whether you want confirmation before using it and whether you want the terminal to be blocked while it is being used.

When specifying any of these issues, you must tell ELinks the situation in which it gets applied, which is typically either the text console or the X window system. So you can specify that you don't want confirmation before running the program when running X, and that you want it otherwise. Similarly, when you are attaching a program (name it foo) to this handler you must tell if it's for X or not (you can attach a second program for the other choice if you want).

**Attaching a program to it** You must tell ELinks the exact command for running it (with any options you wish). In place of the filename you must enter %f. Also the uri can be used as %u.

**Choosing whether you want confirmation before applying it** This is rather simple. If you choose not to do so, then the handler will be automatically called upon when you demand ELinks to show something attached to this handler. Otherwise, if you ask for confirmation, you will be prompted to open it with a external program, with additional choices that include cancelling the operation and viewing it with ELinks. E.g: you may use this option for programs written in C so that you may always choose if you want to read the source with ELinks, or if you want to save it, or if you want to use indent to display it with less.

**Choosing whether to block the terminal** If you don't want to allow going back to ELinks (or any other program from the same terminal) before the external program ends, then you should ask to block the terminal.

### 10.2 Setting up elinks.conf

If you have old configuration files from old ELinks versions or from Links, then you may use the conf-links2elinks.pl script (which is placed at contrib/conv/ on the source distribution) to convert them to the new format. If you don't use it you will need to edit the configuration file, and here you will find how.

#### 10.2.1 Associating a file extension to a MIME type

You can still do this with the UI, from the the Setup->File Extensions submenu.

If you want to do so from the configuration file, enter set mime.extension.ext = "type", replacing ext with the appropriate file extension, and type with its MIME type. E.g. you may want to have set mime.extension.jpg = "image/jpeg".

#### 10.2.2 Defining a handler

For each handler you must define three items, specifying in what context the value of the item should be applied. You must enter set mime.handler.handler-name.item.context = value, replacing handler-name with the name for the handler you are defining, item with the item you are defining for this handler, context with the context this item value is to be applied, and value with the value you want to assign to the item. You must do so for each of the available items: program, ask and block.

The value for program is a string with the exact command you want to be issued to view the file, placing %f were you would put the file name and %u the uri (if needed). The values for ask and block are either 0 (no) or 1 (yes). Available contexts include unix and unix-xwin, which mean UNIX text terminal and X respectively (others can be os2, win32, beos, riscos, ...). The latter does not mean you are running ELinks from X, just that the DISPLAY variable is set so that ELinks may run an X program.

To illustrate it, here is an example. Suppose you want to define the image_viewer handler which should be used to view images. The configuration file may look like this:

```
set mime.handler.image_viewer.unix.ask = 1
set mime.handler.image_viewer.unix-xwin.ask = 0

set mime.handler.image_viewer.unix.block = 1
set mime.handler.image_viewer.unix-xwin.block = 0

set mime.handler.image_viewer.unix.program = "zgv %f"
set mime.handler.image_viewer.unix-xwin.program = "xli %f"
```

In this example the image_viewer handler uses the svgalib image viewer zgv when X is not available, and the X image viewer xli when it is. The terminal would be blocked when X is not available and it would not be when it's available. Finally, ELinks would ask for confirmation before using the handler only with X not available.

#### 10.2.3 Associating a MIME type to a handler

Just enter set mime.type.class.name = "handler", replacing class with the class for the mime type, name with the specific name within that class, and handler with the name for the handler you want to assign to the MIME type. E.g. you may want to have set mime.type.image.jpeg = "image_viewer".

## 11 Managing External Viewers with Mailcap

This document describes the support for Mailcap (RFC 1524) in ELinks. It does not describe the mailcap format. There are plenty of documents on the Web that does this. Google and thou wilt find. ;)

### 11.1 A Short Intro to Mailcap

Mailcap is a file format defined in RFC 1524. Its purpose is to inform multiple mail reading user agent (MUA) programs about the locally-installed facilities for handling mail in various formats. It is designed to work with the Multipurpose Internet Mail Extensions, known as MIME.

ELinks allows MIME handlers to be defined using its own configuration system, so why support mailcap? It can be seen as an alternative or simply as a supplement for setting up MIME handlers in ELinks. Mailcap files are present on most UNIX systems—usually in /etc/mailcap—so this makes it possible for ELinks to know how to handle a great variety of file formats with little configuration. To be able to use mailcap, it has to be compiled into ELinks. This is the default. If you don't need mailcap support, just configure ELinks with the flag: --disable-mailcap.

## 11.2 Parameters to Mailcap Entries

The code has been ported from Mutt and thereby inherits some of its various features and limitation.

The following parameters are supported:

| Parameter | Description |
|---|---|
| %s | The filename that contains the data. |
| %f | The content type, like *text/plain*. |

The following parameters are not supported, since they do not really make much sense for a non-MUA program:

| Parameter | Description |
|---|---|
| %n | The integer number of sub-parts in the multipart |
| %F | The "content-type filename" repeated for each sub-part |
| %{ parameter} | The "parameter" value from the content-type field |

## 11.3 Reading of Mailcap Files

Mailcap files will be read when starting ELinks. The mailcap files to use will be found from the mailcap path, a colon separated list of files similar to the $PATH environment variable. The mailcap path will be determined in the following way:

- From the value of the mime.mailcap.path option in elinks.conf; for example:

```
set mime.mailcap.path = "~/.mailcap:/usr/local/etc/mailcap"
```

- From MAILCAP environment variable.
- If non of the above is defined, the mailcap path defaults to `~/.mailcap:/etc/mailcap`.

## 11.4 Fields

Since mailcap handling is primarily for displaying of resources, all fields like edit, print, compose etc. are ignored.

Note: Test commands are supported, but unfortunately, it's not possible to provide the file when running the test. So any test that requires a file will be considered failed and the handler will not be used.

Unfortunately, there are no *native* support for the copiousoutput field. The field basically mean *needs pager*. So it is handled by appending a pipe and a pager program to the command. The pager program will be read from the `PAGER` environment variable. If this fails, test are made for common pager programs (`/usr/bin/pager`, `/usr/bin/less` and `/usr/bin/more` in that order). So if you define png2ascii as your handler for image/png and specify copiousoutput then the executed command will be "`png2ascii |/usr/bin/less`" if less is your pager or present on your system.

## 11.5 Mailcap Configuration

Apart from the mime.mailcap.path option, you can configure if mailcap support should be disabled. The default being that it is enabled. To disable it just put:

```
set mime.mailcap.enable = 0
```

in elinks.conf.

It is also possible to control whether ELinks should ask you before opening a file. The option is a boolean and can be set like this:

```
set mime.mailcap.ask = 1
```

if you would like to be asked before opening a file.

### 11.6 Some Sample Mailcap Entries

Below are examples of how to specify external viewers:

```
# Use xv if X is running
image/*;                xv %s ; test=test -n "$DISPLAY";

text/x-csrc;            view %s; needsterminal

# Various multimedia files
audio/mpeg;             xmms '%s'; test=test -n "$DISPLAY";
application/pdf;        xpdf '%s'; test=test -n "$DISPLAY";
application/postscript; ps2ascii %s ; copiousoutput
```

## 12 Managing remote ELinks instances

Some programs provide the ability to pass URIs to external programs. When stumbling upon a reference to a page you want to see, it is sometimes a kludge to copy and paste it into ELinks. This is where -remote can be a nifty solution.

When invoking ELinks with the -remote argument, it does not start a new instance, but instead connects to an already running ELinks, making it possible to control that ELinks instance. The -remote command line switch takes a command consisting of the action to invoke and any parameters to the action. Commands must begin with a nonempty sequence of ASCII alphabetic characters followed by optional whitespace and an opening parenthesis. They must end with a closing parenthesis optionally followed by whitespace. Here is an example for opening freshmeat.net in a new tab:

```
$ elinks -remote "openURL(http://freshmeat.net/, new-tab)"
```

When running this command in a terminal, you will see a small delay before ELinks returns. If no running instance was found, it will return with the error message:

```
ELinks: No remote session to connect to.
```

All URLs passed to the openURL() commands can be URL prefixes, so the command above could have simply used "openURL(fm, new-tab)".

### 12.1 Limitations and outstanding issues

Remote control is implemented using the intercommunication socket created in ~/.config/elinks/, so the command has to be run on the same machine as the instance you want to control; or put differently: the two ELinkses need to share a file system that supports socket files, which rules out usage of -remote over NFS. This also implies that the ELinks instance you want to control should be started without passing -no-home nor -no-connect.

The built-in -remote support is to some degree compatible with the one Mozilla provides (http://www.mozilla.org/unix/remote.html), but with some homebrew extensions added and few unsupported features. All the supported actions are documented below.

Under some circumstances, use of the -remote control can cause ELinks to become unresponsive. This is cause by the current key press and mouse focus being redirected to new tabs or dialogs opened by the -remote action.

### 12.2 Remote Actions

The command syntax is case-insensitive. For readability, we use the casing in the listing of supported commands.

`-remote` can also take a list of URLs without an explicit action, in which case the URL arguments will be opened in new tabs in the remote instance. For example, by running:

```
$ elinks -remote slashdot.org fm g:elinks
```

new tabs containing `slashdot.org`, `freshmeat.net` and a Google search of elinks will be opened.

| Command | Description |
|---|---|
| ping() | Checks for existence of a remote instance. Makes it possible for scripts to query if remote controlling is possible. |
| openURL() | Prompts for a URL in current tab by opening the Goto dialog. |
| openURL(URL) | Open the passed URL in current tab. |
| openURL(URL, new-tab) | Opens the passed URL in new tab. |
| openURL(URL, new-window) | Opens the passed URL in new window. |
| xfeDoCommand(openBrowser) | Opens an ELinks instance in a new window. This ELinks instance will connect to the already running one. |

Table 1: Mozilla `-remote` compatible commands.

| Command | Description |
|---|---|
| addBookmark(URL, title) | Bookmarks the passed URL and set title. |
| infoBox(text) | Show text in a message box. |
| reload() | Reload the document in the current tab. |
| search(string) | Search for the string in the current tab |

Table 2: ELinks extensions.

# 13 The tale of ex-mode

Are you a vim-controls nerd who wants to see them everywhere? Welcome.

Actually ELinks doesn't shine in this area yet very much. Heck, the famous hjkl foursome is still occupied by some feeble managers in the default keymap (we have that in our monumental TODO lists). Still, if you know what to touch during the compilation (`--enable-exmode`), you can get at least some familiar reply to the mighty ":" (colon) grip.

## 13.1 What it is

Ex-mode gives you some (still very rough and only marginally complete) access to advanced ELinks commands, to be invoked anywhere anytime, straight and fast.

When you activate the ex-mode (named after the equivalent gadget in the vi text editor flavours), a command line appears at the bottom of the screen for you to type the commands.

Only two kinds of commands are supported so far. First, (almost?) anything that can appear in the configuration file can be used in ex-mode. Second, you can invoke (almost) any action from the ex-mode.

## 13.2 Configuration directives in exmode

There aren't many of these, so we can skim through them fast.

If you want to flip an option you know by name and refuse to engage with the option manager visuals, you can just drop in to the ex-mode and type *set the.option = 1234*. See man elinks.conf (5) or the options manager for the list of options; you can also get a complete options tree saved to elinks.conf if you set *config.saving_style* = 2 (but do **NOT** keep that setting unless you know what are you doing; if we change a default value of some option in future releases, we (generally) know what are we doing - this change won't propagate to you during an upgrade if you already have the original default value saved in your configuration file, though).

It's the same story with keybindings. You can use *bind "main" "h" = "move-cursor-left"*. It's not the same story with keybindings documentation. There is the elinkskeys (5) manual page but it's horribly obsolete, so don't rely on it. You can refer to the keybindings manager for names of actions and even their short descriptions. Also, all the *bind* commands are saved to the configuration file if you set config.saving_style = 2 (but see above).

You can also use *include my.conf*, which will read my.conf as an ELinks configuration file.

Actually, ELinks would eat *#blahblah blah* too, if you see a point in feeding it that kind of stuff.

### 13.3 Actions in exmode - or exmode in action?

There is too many of these, so we should better skim through them fast.

Actually, we already talked about them. It's the last argument to the *bind* command. So, they are those listed in the keybinding manager. So if you enter *move-cursor-left* command, it will move your cursor left - by a single character, making this a little awkward, but it's useful if you sometimes want to easily invoke an action and you don't want to waste a key for it.

Actually, actions could theoretically take arguments too. This is currently implemented only for the *goto-url* action, which can take the location it should go at as a parameter (otherwise it opens the standard well-known dialog as if you pressed *g* in the default keymap).

Regarding the mysterious "(almost)" hinted above, you can never invoke the "quit" action from the exmode - if you type it there, "really-quit" is invoked instead.

### 13.4 How to use it

It's simple. You press *:* (without the apostrophes, of course) and type in the command, then you press enter. E.g., *:set config.saving_style = 3* (this is a good thing), *:quit* (and the game is over). The standard line-editing facility is present (cursor keys and so), and the ex-mode input line has own history.

### 13.5 The "but"s

The biggest usability hurdle so far is that there is no tab-completion. This is why the ex-mode support is not enabled by default and part of the reason why its practical usage is somewhat limited yet - if you don't remember exactly what do you want to invoke, tough beans. Someone shall address this issue in the future.

Also, perhaps wider scale of commands should be implemented in ex-mode. The code is extremely flexible and it is very trivial to make another ex-mode command handler, it's just that no one has done it yet ;-). Also, more actions should be able to take arguments.

## 14 ELinks BitTorrent Client

This chapter provides a small manual for using the implemented BitTorrent client. The BitTorrent client is provided as an optional add-on for ELinks, and needs to be enabled at compile time. To build ELinks with BitTorrent support be sure to either pass `--enable-bittorrent` to `./configure` or change the value of `CONFIG_BITTORRENT` to `yes` in `features.conf`.

### 14.1 Using the BitTorrent Client

To start the client, first go to a site which offers metainfo files. In the following, we will use the site http://www.legaltorrents.com/. Direct ELinks to this site by issuing the command:

```
$ elinks http://www.legaltorrents.com/
```

Use the arrow keys to move between links on the page. Find a link, which points to a metainfo file (a file having the extension `.torrent`). This is indicated in the status bar in the bottom line. Once a metainfo file has been located, press the Return key. This should present you with the following dialog, querying whether the client should start to download the torrent:

```
+--------------------------- What to do? ----------------------------+
|                                                                     |
|  What would you like to do with the file 'go-open-vol-2.torrent'?   |
|  Information about the torrent:                                     |
|  Size: 739 MiB (2959 * 262144 + 161542)                             |
|  Info hash: d85ef7b05288dc49203a7de97545e6c132834011                |
|  Announce URI: http://www.legaltorrents.com:7070/announce           |
```

```
|  Creation date: Jun 26 08:04                                          |
|  Directory: go-open-vol-2                                             |
|                                                                       |
|  [X] 105 MiB go-open-episode-07.mp4                                   |
|  [X] 105 MiB go-open-episode-08.mp4                                   |
|  [X] 105 MiB go-open-episode-09.mp4                                   |
|  [X] 104 MiB go-open-episode-10.mp4                                   |
|  [X] 105 MiB go-open-episode-11.mp4                                   |
|  [X] 107 MiB go-open-episode-12.mp4                                   |
|  [X] 106 MiB go-open-episode-13.mp4                                   |
|  [X]  22 KiB go-open-vol-2.txt                                        |
|                                                                       |
|  [ Download ]  [ Save ]  [ Display ]  [ Show header ]  [ Cancel ]  |
+-----------------------------------------------------------------------+
```

Press the `[ Download ]` button to open the download dialog and start downloading.

Downloaded files can be found in the directory specified by the option `document.download.directory`. There is currently no way to change that.

## 14.2 The Resume Dialog

If you start downloading a torrent which was previously active, the client will first try to resume downloaded data from the disk. The resume progress is shown in the resume dialog, depicted below:

```
+-------------------------------------- Download  ↩
    ---------------------------------------+
|  ↩

    |
| bittorrent:http://www.legaltorrents.com/bit/blue-a-short-film.torrent  ↩
                     |
|  ↩

    |
| [|||||||||||||||||||||||||               26%  ↩
                                           ] |
|  ↩

    |
| Resuming  ↩
                                                                        |
|  ↩

    |
| [ Background ] [ Background with notify ] [ Info ] [ Abort ] [ Abort and delete  ↩
    file ] |
+-----------------------------------------------------------------------------------+
```

Wait until the resume completes. It will automatically launch the download dialog.

## 14.3 The Download Dialog

The download dialog gives an overview of the state of the download, such as progress, and a summary of which places in the torrent pieces have been downloaded from. A view of the download dialog along with highlights of the most important parts of the dialog is given below:


The first bar shows the overall progress of the download along with a percentage of how much data has been downloaded. The second bar provides an overview of the piece completion progress, such as where in the torrent pieces has been downloaded from.

Below the information about download speed and time estimations, several lines about the internal state of the client and its view of the swarm is displayed. For example the `Status` line shows which connection mode the client has entered along with the piece selection strategy if any. Here you will also find information about upload speed and sharing rates. Finally, the `Pieces`

and `Statistics` lines display information from the piece cache. This includes the number of pieces which are currently held in memory, and the number of pieces currently being downloaded.

# 15 Scripting ELinks with Lua

This file documents the Lua scripting interface of the ELinks web browser.

## 15.1 Introduction

### 15.1.1 What is it

Lua scripting capabilities permit users to customize the ELinks behaviour to unusual degree - they allow automatic rewriting of HTML code of the received documents, rewriting of the URLs entered by user etc. You can even write your own bookmarks system with Lua. See also contrib/lua/ for some examples of the possibilities of ELinks Lua support.

Please do not confuse Lua scripting with JavaScript, EcmaScript, VBScript and similar. Those are embedded in page, allowing per-document scripting related to its presentation and providing some degree of interactivity etc. On the contrary, the current Lua support permits scripts to be embedded to the browser directly, changing the behaviour of the browser, not the document.

The original Lua support (in the form of Links-Lua fork of original Links) was written by Peter Wang and Cliff Cunnington. There are some rough edges remaining, but is suitable for everyday use (I have been using it every day for a year).

### 15.1.2 Where to get it

The Lua scripting support comes with the stock ELinks distribution, no additional patches and tweaks should be needed.

The web site of the original Links-Lua is at http://links.sourceforge.net/links-lua/. Some older patches against regular Links are available at http://www.sourceforge.net/projects/links/, but they are not being maintained.

Lua can be found at http://www.lua.org/.

### 15.1.3 What it runs on

The Lua support has only been tested under Linux, although it **should** work under other platforms that ELinks and Lua support (perhaps with some changes to source code?).

Also, note that many of the scripts given here assume a Unix system. Your mileage will definitely vary on other platforms.

## 15.2 Installing

### 15.2.1 Installing Lua

Before you can compile ELinks with Lua support, you must compile and install Lua. The following instructions are for a Linux system. People on other systems should try to enable `popen` support, but this is not necessary (you will lose a bit of functionality though).

1. Download and unpack the Lua `tar.gz` or `zip` somewhere.
2. `cd` into the `lua` directory.
3. Open `config` in a text editor and uncomment the `POPEN` line.
4. Optionally, change the `INSTALL_ROOT line.
5. Run `make; make so; make sobin; make install`.

On systems without shared object support, simply run `make; make install` instead.

Since ELinks 0.11.0, only version 5.0 of Lua is supported. Future versions of ELinks will probably support Lua 5.1 too; see bug 742.

#### 15.2.2 Installing ELinks

Follow the instructions for building ELinks (it is the standard `./configure; make; make install` procedure). During the configure step make sure that Lua has been detected on your system.

#### 15.2.3 Running ELinks with Lua

Simply start ELinks as you normally would. To check you have Lua support compiled in, open up the "Help | About" dialog box. It should list "Scripting (Lua)" under "Features". If not, make sure you do not have other copies of ELinks running, or start ELinks again with the "-no-connect" option on the command-line.

### 15.3 Using ELinks with Lua

Out of the box, ELinks with Lua will do nothing different from regular ELinks. You need to write some scripts.

#### 15.3.1 ELinks Lua additions

The Lua support is based on the idea of **hooks**. A hook is a function that gets called at a particular point during the execution of ELinks. To make ELinks do what you want, you can add and edit such hooks.

The Lua support also adds an extra dialog box, which you can open while in ELinks with the comma (`,`) key. Here you can enter Lua expressions for evaluation, or override it to do something different.

And finally, you can bind keystrokes to Lua functions. These keystrokes won't let you do any more than is possible with the Lua Console, but they're more convenient.

Note that this document assumes you have some knowledge of programming in Lua. For that, you should refer to the Lua reference manual (http://www.lua.org/docs.html). In fact, the language is relatively trivial, though. You could already do wonders with simply refactoring the example scripts.

#### 15.3.2 Config file

On startup, ELinks reads in two Lua scripts. Firstly, a system-wide configuration file called `/etc/elinks/hooks.lua`, then a file in your home directory called `~/.config/elinks/hooks.lua`. From these files, you can include other Lua files with `dofile`, if necessary.

To see what kind of things you should put in here, look at `contrib/lua/hooks.lua`.

#### 15.3.3 Hooks

The following hooks are available.

**goto_url_hook (url, current_url)** This hook is called when the user enters a string into the "Go to URL" dialog box. It is given the string entered, and the current URL (which may be `nil`). It should return a string, which is the URL that ELinks should follow, or `nil` to cancel the operation.

**follow_url_hook (url)** This hook is passed the URL that ELinks is about to follow. It should return a string (the URL modified or unmodified), or `nil` to stop ELinks following the URL

**pre_format_html_hook (url, html)** This hook gets called just before the final time an HTML document is formatted, i.e. it only gets called once, after the entire document is downloaded. It will be passed the URL and HTML text as strings, and should return the modified HTML text, or `nil` if there were no modifications.

**proxy_for_hook (url)** This hook is called when ELinks is about to load a resource from a URL. It should return "PROXY:PORT" (e.g. "localhost:8080") to use the specified proxy, "" to contact the origin server directly, or `nil` to use the default proxy of the protocol.

**lua_console_hook (string)** This hook is passed the string that the user entered into the "Lua Console" dialog box. It should return two values: the type of action to take (`run`, `eval`, `goto-url` or `nil`), and a second argument, which is the shell command to run or the Lua expression to evaluate. Examples:

- `return "run", "someprogram"` will attempt to run the program `someprogram`.
- `return "eval", "somefunction(1+2)"` will attempt to call the Lua function `somefunction` with an argument, 3.
- `return "goto_url", "http://www.bogus.com"` will ask ELinks to visit the URL "http://www.bogus.com".
- `return nil` will do nothing.

**quit_hook ()** This hook is run just before ELinks quits. It is useful for cleaning up things, such as temporary files you have created.

#### 15.3.4 Functions

As well as providing hooks, ELinks provides some functions in addition to the standard Lua functions.

---

**Note**
The standard Lua function `os.setlocale` affects ELinks' idea of the system locale, which ELinks uses for the "System" charset, for the "System" language, and for formatting dates. This may however have to be changed in a future version of ELinks, in order to properly support terminal-specific system locales.

---

**current_url ()** Returns the URL of the current page being shown (in the ELinks session that invoked the function).

**current_link ()** Returns the URL of the currently selected link, or `nil` if none is selected.

**current_title ()** Returns the title of the current page, or `nil` if none.

**current_document ()** Returns the current document as a string, unformatted.

**current_document_formatted ([width])** Returns the current document, formatted for the specified screen width. If the width is not specified, then the document is formatted for the current screen width (i.e. what you see on screen). Note that this function does **not** guarantee all lines will be shorter than `width`, just as some lines may be wider than the screen when viewing documents online.

**pipe_read (command)** Executes `command` and reads in all the data from stdout, until there is no more. This is a hack, because for some reason the standard Lua function `file:read` seems to crash ELinks when used in pipe-reading mode.

**execute (string)** Executes shell commands `string` without waiting for it to exit. Beware that you must not read or write to stdin and stdout. And unlike the standard Lua function `os.execute`, the return value is meaningless.

**tmpname ()** Returns a unique name for a temporary file, or `nil` if no such name is available. The returned string includes the directory name. Unlike the standard Lua function `os.tmpname`, this one generates ELinks-related names (currently with "elinks" at the beginning of the name).

---

**Warning**
The `tmpname` function creates the file but does not guarantee exclusive access to it: another process may delete the file and recreate it. This exposes you to symlink attacks by other users. To avoid the risk, use `io.tmpfile` instead; unfortunately, it does not tell you the name of the file.

---

**bind_key (keymap, keystroke, function)** Currently, `keymap` must be the string `"main"`. Keystroke is a keystroke as you would write it in the ELinks config file `~/.config/elinks/elinks.conf`. The function `function` should take no arguments, and should return the same values as `lua_console_hook`.

**edit_bookmark_dialog (cat, name, url, function)** Displays a dialog for editing a bookmark, and returns without waiting for the user to close the dialog. The return value is `1` if successful, `nil` if arguments are invalid, or nothing at all if out of memory. The first three arguments must be strings, and the user can then edit them in input fields. There are also *OK* and *Cancel* buttons in the dialog. If the user presses *OK*, ELinks calls `function` with the three edited strings as arguments, and it should return similar values as in `lua_console_hook`.

**xdialog (string [, more strings...], function)** Displays a generic dialog for editing multiple strings, and returns without waiting for the user to close the dialog. The return value is `1` if successful, `nil` if arguments are invalid, or nothing at all if out of memory. All arguments except the last one must be strings, and ELinks places them in input fields in the dialog. There can be at most 5 such strings. There are also *OK* and *Cancel* buttons in the dialog. If the user presses *OK*, ELinks calls `function` with the edited strings as arguments, and it should return similar values as in `lua_console_hook`.

**set_option (option, value)** Sets an ELinks option. The first argument `option` must be the name of the option as a string. ELinks then tries to convert the second argument `value` to match the type of the option. If successful, `set_option` returns `value`, else `nil`.

**get_option (option)** Returns the value of an ELinks option. The argument `option` must be the name of the option as a string. If the option does not exist, `get_option` returns `nil`.

#### 15.3.5 Variables

**elinks_home** The name of the ELinks home directory, as a string. Typically this is the .elinks subdirectory of the user's home directory.

#### 15.3.6 User protocol

There is one more little thing which Links-Lua adds, which will not be described in detail here. It is the fake "user:" protocol, which can be used when writing your own addons. It allows you to generate web pages containing links to "user://blahblah", which can be intercepted by the `follow_url_hook` (among other things) to perform unusual actions. For a concrete example, see the bookmark addon.

### 15.4 Example recipes

This chapter contains some example scripts that you can use. All of them come from `contrib/lua/hooks.lua`. I really recommend you to see it directly instead of copying code out of this document. Also, not everything in there is covered here.

If you would like to contribute scripts, that would be great! Please send them to me at tjaden@users.sourceforge.net. Cliff and I plan to start a script repository, provided we get some contributions. As for script ideas, you'll just have to be a little creative :-)

Also take a look at the `contrib/lua/` directory in the ELinks distribution. Note that Peter and Cliff don't maintain the Lua support intensively anymore, thus it would be probably nice to Cc me (pasky@ucw.cz) if you want to contribute some patch, so that I would be able to add it to the ELinks distribution.

#### 15.4.1 Go to URL on steroids

There are some web sites that I visit often. Bookmarks are okay, but they are separate from the "Go to URL" dialog box, so I keep forgetting to use them. Also, when I visit a search engine home page, all I really want to do is enter a search term.

The following script allows me to type certain strings into the "Go to URL" dialog box, and it will convert them to the URL I actually want to visit. As a bonus, it allows me perform some searches on sites like Google without loading up the front page first.

---

**Tip**
The "URI rewriting" feature of ELinks handles many of the same tasks as the Lua hook shown here, and you can conveniently configure it via the option manager. It is not quite as versatile, though.

---

```
function match (prefix, url)
    return string.sub (url, 1, string.len (prefix)) == prefix
end

function strip (str)
    return string.gsub (str, "^%s*(.-)%s*$", "%1")
end

function plusify (str)
    return string.gsub (str, "%s", "+")
end

function goto_url_hook (url, current_url)
    -- Google search (e.g. ,gg unix browsers).
    if match (",gg", url) then
        url = plusify (strip (string.sub (url, 4)))
        return "http://www.google.com/search?q="..url.."&btnG=Google+Search"

    -- Freshmeat search.
    elseif match (",fm", url) then
        url = plusify (strip (string.sub (url, 4)))
        return "http://www.freshmeat.net/search/?q="..url

    -- Dictionary.com search (e.g. ,dict congenial).
    elseif match (",dict", url) then
        url = plusify (strip (string.sub (url, 6)))
        return "http://www.dictionary.com/cgi-bin/dict.pl?db=%2A&term="..url

    -- RPM search (e.g. ,rpm links).
    elseif match (",rpm", url) then
        url = plusify (strip (string.sub (url, 5)))
        return "http://www.rpmfind.net/linux/rpm2html/search.php?query="
                ..url.."&submit=Search+..."

    -- Netcraft.com search (e.g. ,whatis www.google.com).
    elseif match (",whatis", url) then
        url = plusify (strip (string.sub (url, 8)))
        return "http://uptime.netcraft.com/up/graph/?host="..url

    -- LinuxToday home page.
    elseif match (",lt", url) then
        return "http://linuxtoday.com/"

    -- Weather forecast for Melbourne, Australia.
    elseif match (",forecast", url) then
        return "http://www.bom.gov.au/cgi-bin/wrap_fwo.pl?IDV10450.txt"

    -- Unmatched
    else
        return url
    end
end
```

#### 15.4.2 Expanding ~ (tilde)

By adding an extra snippet of code to the previous example, we can make ELinks expand pathnames such as `~/foo/bar` and `~user/zappo`, like in the shell and other Unix programs.

```
function goto_url_hook (url, current_url)
                .
```

```
                .

    -- Expand ~ to home directories.
    elseif match ("~", url) then
        if string.sub(url, 2, 2) == "/" then    -- ~/foo
            return os.getenv ("HOME")..string.sub(url, 2)
        else                                    -- ~foo/bar
            return "/home/"..string.sub(url, 2)
        end

                .
                .
```

#### 15.4.3 Filtering crap

Many web pages nowadays have columns to the left and right of the text, which are utterly useless. If you happen to be viewing the page in a 80x25 screen, the text you want to read ends up crammed into a tiny space in the centre. We use ELinks Lua support to manipulate the HTML before it reaches the parser.

##### 15.4.3.1 linuxtoday.com

**Note**
This recipe is out of date for the web site.

Linux Today has two problems when viewed in ELinks: the useless columns on the left and the right and all the text appears in cyan. Here is a quick recipe to fix that:

```
-- Plain string.find (no metacharacters)
function sstrfind (s, pattern)
    return string.find (s, pattern, 1, true)
end

function pre_format_html_hook (url, html)
    -- Strip the left and right columns from Linux Today pages
    -- and change the font colour to white.
    if sstrfind (url, "linuxtoday.com") then
        if sstrfind (url, "news_story") then
            html = string.gsub (html, '<TABLE CELLSPACING="0".-</TABLE>', '', 1)
            html = string.gsub (html, '<TR BGCOLOR="#FFF.-</TR></TABLE>', '', 1)
        else
            html = string.gsub (html, 'WIDTH="120">\n<TR.+</TABLE></TD>', '>', 1)
        end
        html = string.gsub (html, '<A HREF="http://www.internet.com.-</A>', '')
        html = string.gsub (html, "<IFRAME.-</IFRAME>", "")
        -- emphasis in text is lost
        return string.gsub (html, 'text="#002244"', 'text="#001133"', 1)
    end

    return nil
end
```

##### 15.4.3.2 linuxgames.com

**Note**
This recipe is out of date for the web site.

Here is a simpler example, for http://www.linuxgames.com/.

```
function pre_format_html_hook (url, html)
                .
                .

    elseif string.find (url, "linuxgames.com", 1, true) then
        return string.gsub (html, "<CENTER>.-</center>", "", 1)

                .
                .
```

#### 15.4.4 Reading gzipped files

**Note**
ELinks already supports gzipped files natively.

Sometimes documents come gzipped in order to save space, but then you need to uncompress them to read them with ELinks. Here is a recipe to handle gzipped files on a Unix system.

**Warning**
This recipe opens a temporary file insecurely.

```
function pre_format_html_hook (url, html)
                .
                .

    -- Handle gzip'd files within reasonable size.
    if string.find (url, "%.gz$") and string.len (html) < 65536 then
        local name = tmpname ()
        local file = io.open (name, "wb")
        file:write (html)
        file:close ()
        html = pipe_read ("(gzip -dc "..name.." || cat "..name..") 2>/dev/null")
        os.remove (name)
        return html
    end

                .
                .
```

#### 15.4.5 Printing

Printing a web page with ELinks usually involves quite a few steps: Save the current document onto disk. Run it through ELinks on the command-line (so it fits into 80 columns) to generate a plain text version. Remove the 80th column from the text version, as it will make printers wrap down to the next line. Finally, run the processed file through `lpr', then delete it.

The following functions allow you to print web pages directly from ELinks, using `lpr' or `enscript'. Type `lpr()` or `enscript()` in the Lua Console to run them. (In the `hooks.lua`, I have also made it so you can just type `lpr` or `enscript`.)

**Note**
The `io.popen` function is not available on all platforms.

```
function pipe_formatted_to (program)
    local lp, errmsg = io.popen (program, "w")
    if lp == nil then
        error (errmsg)
    else
        lp:write (current_document_formatted (79))
        lp:close ()
    end
end

-- Send the current document to 'lpr'.
function lpr ()
    pipe_formatted_to ("lpr")
end

-- Send the current document to 'enscript'.
function enscript ()
    pipe_formatted_to ("enscript -fCourier8")
end
```

#### 15.4.6 Deferring to Netscape

If you come across a brain-dead web page that is totally unreadable with ELinks, you'd probably want to open it with a graphical browser. The following function opens the current document in Netscape.

**Tip**
You can also use the built-in "URI passing" feature for this.

```
-- When starting Netscape: Set to 'nil' if you do not want
-- to open a new window for each document.
netscape_new_window = 1

-- Open current document in Netscape.
function netscape ()
    local new = netscape_new_window and ",new_window" or ""
    execute ("( netscape -remote 'openURL("..current_url ()..new..")'"
             .." || netscape '"..current_url ()..."' ) 2>/dev/null &")
end
```

#### 15.4.7 Alternative bookmark system

Many people would like to have a bookmark system with categories (note that ELinks already supports that, marketing name Hierarchical bookmarks), and also to be able to view them and search for them in an HTML page. I have written an alternative bookmark system (for ELinks), which some people may like better than the standard bookmark system.

#### 15.4.8 More ideas

- The Lua interface needs to be redesigned to provide more flexible, coherent and usable interface to the scripts.

- Cliff Cunnington had a neat idea of clipping text that you see in web pages (you enter a regexp that will match the start and end of the text you want to clip), and saving the text to disk, along with the URL and timestamp. This would help if you find that you can't ever remember where you had seen a piece of text, or if you want to keep a piece of information but don't need to save the entire page.
- People who use download management programs could write a function to send the current link to their favourite downloading program.
- If you wrote a small C program to put text into the X11 selection clipboard, you could pass the current link or URL to that program, to make it easier to paste URLs into other windows. It might be possible to do the same with GPM, or the KDE/GNOME equivalents.
- Send the current page to Babelfish for translation.
- Look for stupid JavaScript URLs and convert them to something usable.
- More things are possible, I'm sure. If you have an idea that requires another hook or function, contact me (Peter Wang) and I'll see what I can do.

# 16 Scripting ELinks with ECMAScript

As a user of ELinks, you can control its behaviour by writing scripts in ECMAScript. Unlike scripts in SCRIPT elements of HTML, these user scripts run with all the permissions of your user account, the same as with Lua. The object model is very different too.

Support for ECMAScript user scripts was first added in ELinks 0.11.0. The `configure` script enables it by default if the required SpiderMonkey library has been installed, but you can disable it with `configure --disable-sm-scripting` or by editing features.conf.

**Warning**
ECMAScript scripting is still a bit experimental: there seem to be ways to crash ELinks with it, and the object model may change. However, if you don't have a `hooks.js` file, there is not much risk in enabling the feature at compile time.

When ELinks starts up, it evaluates the ECMAScript file `hooks.js` in your ELinks configuration directory (thus normally `~/.config/elinks/hooks.js` on Unix-like systems), or if the file does not exist there, then in the system-wide ELinks configuration directory (the location depends on how ELinks was built, but `/etc/elinks/hooks.js` is typical).

In the ELinks source tree, the `contrib/smjs` directory contains some examples about scripting ELinks with ECMAScript. Please see the `README` file in that directory for details.

## 16.1 Global Object

The global object provided to ECMAScript user scripts contains the standard ECMAScript classes, as well as the following:

### 16.1.1 Global Object Methods

**do_file(path)** Load and evaluate the file with the given path (string). For example:

```
do_file("/home/me/.config/elinks/hooks.js");
```

will reload your hooks file.

**Compatibility:** ELinks 0.11.0

### 16.1.2 Global Object Properties

**elinks (elinks)** A reference to the ELinks object.

**Compatibility:** ELinks 0.11.0

## 16.2 ELinks Object

The global elinks property refers to this object.

### 16.2.1 ELinks Object Methods

**elinks.alert(message)** Display the given message (string) in a message box. For example:

```
elinks.alert("Hello, world!");
```

will display a friendly greeting.

**Compatibility:** ELinks 0.11.0

**elinks.execute(command)** Execute the given command (string) on the current terminal. For example:

```
var quoted_uri = "'" + elinks.location.replace(/'/g, "'\\''") + "'";
elinks.execute("firefox " + quoted_uri);
```

will run Firefox with the URI of the current document.

**Compatibility:** ELinks 0.12pre1

**Warning**
One must be very careful with *elinks.execute*, because depending on the OS, the command may be subject to interpretation by a command shell language. When constructing the command string, be sure to quote any dubious parts (such as the URI of the current document, as above).

**elinks.load_uri(uri, callback)** Load the given URI (string). When the URI completes loading, ELinks calls the given callback (function). The callback is passed the cache object that corresponds to the URI. For example:

```
elinks.load_uri("http://www.eldar.org/cgi-bin/fortune.pl?text_format=yes",
                function (cached) { elinks.alert(cached.content); });
```

displays a fortune.

The cache object will not expire until after the callback returns.

**Compatibility:** ELinks 0.12pre1

### 16.2.2 ELinks Object Properties

**elinks.home (string)** ELinks's "home" directory, where it stores its configuration files. Read-only. For example,

```
do_file(elinks.home + "hooks.js");
```

will reload your hooks file.

**Compatibility:** ELinks 0.11.0

**elinks.location (string)** The URI of the currently open document. This can be read to get a string with the URI or set to load a different document. For example,

```
elinks.location = elinks.location + "/..";
```

will go up a directory (if the URI doesn't end in a file).

**Compatibility:** ELinks 0.11.0

**elinks.bookmarks (hash)** This is a hash, the elements of which correspond to the bookmarks. One can delve into the bookmarks hierarchy in a reasonably nifty fashion, just by using standard ECMAScript syntax:

```
elinks.bookmarks.x.children.y.children.z.children.foo.title
```

gets the title of the bookmark titled "foo" under the folder "z", which is a subfolder of "y", which is a subfolder of "x".

**Compatibility:** ELinks 0.11.0

A bookmark object has these properties:

**item.title (string)** This is the title of the bookmark. It can be read and set.

**item.url (string)** This is the URI of the bookmark. It can be read and set.

**item.children (hash)** This is a hash, the elements of which are the bookmarks that are children to the item. It is read-only.

**elinks.globhist (hash)** This is a hash, the elements of which correspond to entries in ELinks's global history. The hash is indexed by URI. For example,

```
elinks.globhist["file:///"]
```

will get you the history item for your root directory.

**Compatibility:** ELinks 0.12pre1

A history item has these properties:

**item.title (string)** This is the title of the history item. It can be read and set.

**item.url (string)** This is the URI of the history item. It can be read and set.

**item.last_visit (number)** This is the UNIX time of the last visit time for the item. UNIX time is the number of seconds that have passed between the UNIX epoch (which is 1970-01-01 00:00:00 UTC) and the represented time. Note that this is *seconds* since the epoch, whereas ECMAScript likes to use *milliseconds* since the epoch. This property can be set or read.

**elinks.action (hash)** This hash lets you call the built-in actions of ELinks. For example, you can call `elinks.action.auth_mana` to open the authentication manager. The names of the actions are the same as in elinks.conf or in the keybinding manager, except they have underscores instead of dashes in order to make them valid ECMAScript identifiers.

**Compatibility:** ELinks 0.12pre1

---

**Note**

When you read an action function from this hash, ELinks binds it to the current tab; any later calls to the function throw errors if that tab no longer has the focus (in its terminal). This may be changed in a future version. It is safest to call the function right away, rather than save it in a variable and call it later.

---

**elinks.keymaps (hash)** This is a hash, the elements of which correspond to ELinks's keymaps. Currently, there are three: *elinks.keymaps.main*, *elinks.keymaps.edit*, and *elinks.keymaps.menu*. These elements are also hashes, the elements of which correspond to bindings. For example, `elinks.keymaps.main["q"]` is the binding to the "q" key in the main map. These bindings can be read, to get the name of the action to which the key is bound, or set to one of:

- A string with the name of the ELinks action.
- A function, which will thenceforth be called when the key is pressed.

- The string `"none"`, to unbind the key. You can also use the `null` value for this purpose, but that crashes ELinks 0.11.4 and 0.12pre1 (bug 1027), so it may be best to use the string for now.

For example,

```
elinks.keymaps.main["!"] = function () { elinks.alert("Hello!"); }
```

binds the "!" key in the main map to a function that displays a friendly alert.

```
elinks.keymaps.main["/"] = "search-typeahead-text";
```

changes the "/" key to use the nice typeahead search function instead of opening that ugly old search dialogue box.

**Compatibility:** ELinks 0.11.0, unless you use `null`.

---

**Note**

Do not read a function from elinks.action, e.g. `elinks.action.search_typeahead_text`, and place it in a keymap. ELinks binds such functions to the current tab when the script reads them from *elinks.action*, so they will not work right in other tabs. Use the name of the action instead.

---

**elinks.vs (view_state)** This property refers to the view-state object for the current document, if any.

**Compatibility:** ELinks 0.12pre1

#### 16.2.3 ELinks Object Hooks

These are actually properties, but a special case: one assigns functions to them, which functions are called at certain events.

In the ELinks source tree, `contrib/smjs/hooks.js` provides a mechanism with which multiple scripts can add their functions to the same hooks. Please see `contrib/smjs/README` for details.

**elinks.preformat_html(cached, vs)** This function is called every time a document is loaded, before the document is actually rendered, to give scripts the opportunity to modify it. The first parameter is the cache object and the second is the view-state object.

The cache object will not expire until after this function returns.

**Compatibility:** ELinks 0.11.1 as described. ELinks 0.11.0 did not provide the *vs* argument.

**elinks.goto_url_hook(url)** This function is called every time the user enters something in the *Go to URL* box. The url (string) can be modified or not, and the returned string is substituted for what the user entered. If the value `false` is returned, the URL is not changed and further hooks in ELinks are not run.

**Compatibility:** ELinks 0.11.0

**elinks.follow_url_hook(url)** This function is called every time the user tries to load a document, whether by following a link, by entering a URI in the Go to URL box, by setting elinks.location, or whatever. It behaves the same as elinks.goto_url_hook above.

**Compatibility:** ELinks 0.11.0

### 16.3 Cache Object

The cache object mentioned in the descriptions of elinks.load_uri and elinks.preformat_html is a wrapper for the internal ELinks cache object. ELinks passes the ECMAScript cache object as an argument to your ECMAScript function, and keeps the corresponding document in the cache until the function returns. After that, ELinks may remove the document from the cache, even if the function has saved the cache object to some global variable. Such an expired cache object does not work but it does not crash ELinks either.

**Compatibility:** ELinks 0.11.0

### 16.3.1 Cache Object Properties

**cached.content (string)** This is the content received from the server. It can be read and set.

**cached.type (string)** This is the MIME type of the cache entry. It can be read and set.

**cached.length (number)** This is the length of cached.content. It is read-only.

**cached.head (string)** This is the header received from the server. It can be read and set.

**cached.uri (string)** This is the URI of the cache entry. It is read-only.

## 16.4 View-state Object

The view-state object mentioned in the descriptions of elinks.preformat_html and elinks.vs is a wrapper for the internal ELinks view_state object. The view state holds information on how the current document is being displayed.

**Compatibility:** ELinks 0.11.1

### 16.4.1 View-state Object Properties

**vs.plain (boolean)** Whether the current document is rendered as HTML or displayed as plaintext. This can be read and set.

**vs.uri (string)** This is the URI of the current document. It is read-only.