Privoxy 3.0.8 User Manual

$Id: user-manual.sgml,v 2.55 2008/01/19 21:26:37 hal9 Exp $

The *Privoxy User Manual* gives users information on how to install, configure and use Privoxy.

Privoxy is a non-caching web proxy with advanced filtering capabilities for enhancing privacy, modifying web page data, managing HTTP cookies, controlling access, and removing ads, banners, pop-ups and other obnoxious Internet junk. Privoxy has a flexible configuration and can be customized to suit individual needs and tastes. Privoxy has application for both stand-alone systems and multi-user networks.

Privoxy is based on Internet Junkbuster (tm).

You can find the latest version of the *Privoxy User Manual* at http://www.privoxy.org/user-manual/. Please see the Contact section on how to contact the developers.

Table of Contents

Table of Contents

1. Introduction

This documentation is included with the current stable version of Privoxy, v.3.0.8.

1.1. Features

In addition to the core features of ad blocking and cookie management, Privoxy provides many supplemental features, that give the end-user more control, more privacy and more freedom:

- Can be run as an "intercepting" proxy, which obviates the need to configure browsers individually.
- Sophisticated actions and filters for manipulating both server and client headers.
- Can be chained with other proxies.
- Integrated browser based configuration and control utility at http://config.privoxy.org/ (shortcut: http://p.p/). Browser-based tracing of rule and filter effects. Remote toggling.
- Web page filtering (text replacements, removes banners based on size, invisible "web-bugs", JavaScript and HTML annoyances, pop-up windows, etc.)
- Modularized configuration that allows for standard settings and user settings to reside in separate files, so that installing updated actions files won't overwrite individual user settings.
- Support for Perl Compatible Regular Expressions in the configuration files, and a more sophisticated and flexible configuration syntax.
- Improved cookie management features (e.g. session based cookies).
- GIF de-animation.
- Bypass many click-tracking scripts (avoids script redirection).
- Multi-threaded (POSIX and native threads).
- User-customizable HTML templates for all proxy-generated pages (e.g. "blocked" page).
- Auto-detection and re-reading of config file changes.
- Improved signal handling, and a true daemon mode (Unix).
- Every feature now controllable on a per-site or per-location basis, configuration more powerful and versatile over-all.
- Many smaller new features added, limitations and bugs removed, and security holes fixed.

2. Installation

Privoxy is available both in convenient pre-compiled packages for a wide range of operating systems, and as raw source code. For most users, we recommend using the packages, which can be downloaded from our Privoxy Project Page.

Note: On some platforms, the installer may remove previously installed versions, if found. (See below for your platform). In any case *be sure to backup your old configuration if it is valuable to you.* See the note to upgraders section below.

2.1. Binary Packages

How to install the binary packages depends on your operating system:

2.1.1. Red Hat and Fedora RPMs

RPMs can be installed with `rpm -Uvh privoxy-3.0.8-1.rpm`, and will use `/etc/privoxy` for the location of configuration files.

Note that on Red Hat, Privoxy will *not* be automatically started on system boot. You will need to enable that using **chkconfig**, **ntsysv**, or similar methods.

If you have problems with failed dependencies, try rebuilding the SRC RPM: `rpm --rebuild privoxy-3.0.8-1.src.rpm`. This will use your locally installed libraries and RPM version.

Also note that if you have a Junkbuster RPM installed on your system, you need to remove it first, because the packages conflict. Otherwise, RPM will try to remove Junkbuster automatically if found, before installing Privoxy.

2.1.2. Debian and Ubuntu

DEBs can be installed with `apt-get install privoxy`, and will use `/etc/privoxy` for the location of configuration files.

2.1.3. Windows

Just double-click the installer, which will guide you through the installation process. You will find the configuration files in the same directory as you installed Privoxy in.

Version 3.0.5 beta introduced full Windows service functionality. On Windows only, the Privoxy program has two new command line arguments to install and uninstall Privoxy as a *service*.

Arguments:

`--install`[:*`service_name`*]

`--uninstall`[:*`service_name`*]

After invoking Privoxy with **--install**, you will need to bring up the Windows service console to assign the user you want Privoxy to run under, and whether or not you want it to run whenever the system starts. You can start the Windows services console with the following command: **services.msc**. If you do not take the manual step of modifying Privoxy's service settings, it will not start. Note too that you will need to give Privoxy a user account that actually exists, or it will not be permitted to write to its log and configuration files.

2.1.4. Solaris

Create a new directory, `cd` to it, then unzip and untar the archive. For the most part, you'll have to figure out where things go.

2.1.5. OS/2

First, make sure that no previous installations of Junkbuster and / or Privoxy are left on your system. Check that no Junkbuster or Privoxy objects are in your startup folder.

Then, just double-click the WarpIN self-installing archive, which will guide you through the installation process. A shadow of the Privoxy executable will be placed in your startup folder so it will start automatically whenever OS/2 starts.

The directory you choose to install Privoxy into will contain all of the configuration files.

2.1.6. Mac OSX

Unzip the downloaded file (you can either double-click on the file from the finder, or from the desktop if you downloaded it there). Then, double-click on the package installer icon named `Privoxy.pkg` and follow the installation process. Privoxy will be installed in the folder `/Library/Privoxy`. It will start automatically whenever you start up. To prevent it from starting automatically, remove or rename the

folder `/Library/StartupItems/Privoxy`.

To start Privoxy by hand, double-click on `StartPrivoxy.command` in the `/Library/Privoxy` folder. Or, type this command in the Terminal:

```
/Library/Privoxy/StartPrivoxy.command
```

You will be prompted for the administrator password.

2.1.7. AmigaOS

Copy and then unpack the `lha` archive to a suitable location. All necessary files will be installed into Privoxy directory, including all configuration and log files. To uninstall, just remove this directory.

2.1.8. FreeBSD

Privoxy is part of FreeBSD's Ports Collection, you can build and install it with `cd /usr/ports/www/privoxy; make install clean`.

If you don't use the ports, you can fetch and install the package with `pkg_add -r privoxy`.

The port skeleton and the package can also be downloaded from the File Release Page, but there's no reason to use them unless you're interested in the beta releases which are only available there.

2.1.9. Gentoo

Gentoo source packages (Ebuilds) for Privoxy are contained in the Gentoo Portage Tree (they are not on the download page, but there is a Gentoo section, where you can see when a new Privoxy Version is added to the Portage Tree).

Before installing Privoxy under Gentoo just do first `emerge rsync` to get the latest changes from the Portage tree. With `emerge privoxy` you install the latest version.

Configuration files are in `/etc/privoxy`, the documentation is in `/usr/share/doc/privoxy-3.0.8` and the Log directory is in `/var/log/privoxy`.

2.2. Building from Source

The most convenient way to obtain the Privoxy sources is to download the source tarball from our project download page.

If you like to live on the bleeding edge and are not afraid of using possibly unstable development versions, you can check out the up-to-the-minute version directly from the CVS repository.

To build Privoxy from source, autoconf, GNU make (gmake), and, of course, a C compiler like gcc are required.

When building from a source tarball, first unpack the source:

```
tar xzvf privoxy-3.0.8-src* [.tgz or .tar.gz]
cd privoxy-3.0.8
```

For retrieving the current CVS sources, you'll need a CVS client installed. Note that sources from CVS are typically development quality, and may not be stable, or well tested. To download CVS source, check the Sourceforge documentation, which might give commands like:

```
cvs -d:pserver:anonymous@ijbswa.cvs.sourceforge.net:/cvsroot/ijbswa login
cvs -z3 -d:pserver:anonymous@ijbswa.cvs.sourceforge.net:/cvsroot/ijbswa co current
cd current
```

This will create a directory named `current/`, which will contain the source tree.

You can also check out any Privoxy "branch", just exchange the current name with the wanted branch name (Example: v_3_0_branch for the 3.0 cvs tree).

It is also strongly recommended to not run Privoxy as root. You should configure/install/run Privoxy as an unprivileged user, preferably by creating a "privoxy" user and group just for this purpose. See your local documentation for the correct command line to do add new users and groups (something like **adduser**, but the command syntax may vary from platform to platform).

`/etc/passwd` might then look like:

```
privoxy:*:7777:7777:privoxy proxy:/no/home:/no/shell
```

And then `/etc/group`, like:

```
privoxy:*:7777:
```

Some binary packages may do this for you.

Then, to build from either unpacked tarball or CVS source:

```
autoheader
autoconf
./configure      # (--help to see options)
make             # (the make from GNU, sometimes called gmake)
su               # Possibly required
make -n install  # (to see where all the files will go)
make -s install  # (to really install, -s to silence output)
```

Using GNU **make**, you can have the first four steps automatically done for you by just typing:

```
make
```

in the freshly downloaded or unpacked source directory.

To build an executable with security enhanced features so that users cannot easily bypass the proxy (e.g. "Go There Anyway"), or alter their own configurations, **configure** like this:

```
./configure  --disable-toggle  --disable-editor  --disable-force
```

Then build as above. In Privoxy 3.0.7 and later, all of these options can also be disabled through the configuration file.

WARNING: If installing as root, the install will fail unless a non-root user or group is specified, or a `privoxy` user and group already exist on the system. If a non-root user is specified, and no group, then the installation will try to also use a group of the same name as "user". If a group is specified (and no user), then the support files will be installed as writable by that group, and owned by the user running the installation.

configure accepts `--with-user` and `--with-group` options for setting user and group ownership of the configuration files (which need to be writable by the daemon). The specified *user must already exist*. When starting Privoxy, it must be run as this same user to insure write access to configuration and log files!

Alternately, you can specify `user` and `group` on the **make** command line, but be sure both already exist:

```
make -s install  USER=privoxy GROUP=privoxy
```

The default installation path for **make install** is `/usr/local`. This may of course be customized with the various **./configure** path options. If you are doing an install to anywhere besides `/usr/local`, be sure to set the appropriate paths with the correct configure options (**./configure --help**). Non-privileged users must of course have write access permissions to wherever the target installation is going.

If you do install to `/usr/local`, the install will use `sysconfdir=$prefix/etc/privoxy` by default. All other destinations, and the direct usage of `--sysconfdir` flag behave like normal, i.e. will not add the extra `privoxy` directory. This is for a safer install, as there may already exist another program that uses a file with the "config" name, and thus makes `/usr/local/etc` cleaner.

If installing to `/usr/local`, the documentation will go by default to `$prefix/share/doc`. But if this directory doesn't exist, it will then try `$prefix/doc` and install there before creating a new `$prefix/share/doc` just for Privoxy.

Again, if the installs goes to `/usr/local`, the `localstatedir` (ie: `var/`) will default to `/var` instead of `$prefix/var` so the logs will go to `/var/log/privoxy/`, and the pid file will be created in `/var/run/privoxy.pid`.

make install will attempt to set the correct values in `config` (main configuration file). You should check this to make sure all values are correct. If appropriate, an init script will be installed, but it is up to the user to determine how and where to start Privoxy. The init script should be checked for correct paths and values, if anything other than a default install is done.

If install finds previous versions of local configuration files, most of these will not be overwritten, and the new ones will be installed with a "new" extension. default.action, default.filter, and standard.action *will be overwritten*. You will then need to manually update the other installed configuration files as needed. The default template files *will* be overwritten. If you have customized, local templates, these should be stored safely in a separate directory and defined in `config` by the "templdir" directive. It is of course wise to always back-up any important configuration files "just in case". If a previous version of Privoxy is already running, you will have to restart it manually.

For more detailed instructions on how to build Redhat RPMs, Windows self-extracting installers, building on platforms with special requirements etc, please consult the developer manual.

2.3. Keeping your Installation Up-to-Date

As user feedback comes in and development continues, we will make updated versions of both the main actions file (as a separate package) and the software itself (including the actions file) available for download.

If you wish to receive an email notification whenever we release updates of Privoxy or the actions file, subscribe to our announce mailing list, ijbswa-announce@lists.sourceforge.net.

In order not to lose your personal changes and adjustments when updating to the latest `default.action` file we *strongly recommend* that you use `user.action` and `user.filter` for your local customizations of Privoxy. See the Chapter on actions files for details.

3. What's New in this Release

There are many improvements and new features since Privoxy 3.0.6, the last stable release:

- Two new actions server-header-tagger and client-header-tagger that can be used to create arbitrary "tags" based on client and server headers. These "tags" can then subsequently be used to control the other actions used for the current request, greatly increasing Privoxy's flexibility and selectivity. See tag patterns for more information on tags.
- Header filtering is done with dedicated header filters now. As a result the actions "filter-client-headers" and "filter-server-headers" that were introduced with Privoxy 3.0.5 to apply content filters to the headers have been removed. See the new actions server-header-filter and client-header-filter for details.
- There are four new options for the main `config` file:
    - allow-cgi-request-crunching which allows requests for Privoxy's internal CGI pages to be blocked, redirected or (un)trusted like ordinary requests.
    - split-large-forms that will work around a browser bug that caused IE6 and IE7 to ignore the Submit button on the Privoxy's edit-actions-for-url CGI page.
    - accept-intercepted-requests which allows to combine Privoxy with any packet filter to create an intercepting proxy for HTTP/1.1 requests (and for HTTP/1.0 requests with Host header set). This means clients can be forced to use Privoxy even if their proxy settings are configured differently.
    - templdir to designate an alternate location for Privoxy's locally customized CGI templates so that these are not overwritten during upgrades.
- A new command line option `--pre-chroot-nslookup hostname` to initialize the resolver library before chroot'ing. On some systems this reduces the number of files that must be copied into the chroot tree. (Patch provided by Stephen Gildea)
- The forward-override action allows changing of the forwarding settings through the actions files. Combined with tags, this allows to choose the forwarder based on client headers like the `User-Agent`, or the request origin.
- The redirect action can now use regular expression substitutions against the original URL.
- zlib support is now available as a compile time option to filter compressed content. Patch provided by Wil Mahan.
- Improve various filters, and add new ones.
- Include support for RFC 3253 so that `Subversion` works with Privoxy. Patch provided by Petr Kadlec.
- Logging can be completely turned off by not specifying a logfile directive.
- A number of improvements to Privoxy's internal CGI pages, including the use of favicons for error and control pages.
- Many bugfixes, memory leaks addressed, code improvements, and logging improvements.

For a more detailed list of changes please have a look at the ChangeLog.

3.1. Note to Upgraders

A quick list of things to be aware of before upgrading from earlier versions of Privoxy:

- The recommended way to upgrade Privoxy is to backup your old configuration files, install the new ones, verify that Privoxy is working correctly and finally merge back your changes using diff and maybe patch.

  There are a number of new features in each Privoxy release and most of them have to be explicitly enabled in the configuration files. Old configuration files obviously don't do that and due to syntax changes using old configuration files with a new Privoxy isn't always possible anyway.
- Note that some installers remove earlier versions completely, including configuration files, therefore you should really save any important configuration files!
- On the other hand, other installers don't overwrite existing configuration files, thinking you will want to do that yourself.
- `standard.action` now only includes the enabled actions. Not all actions as before.
- In the default configuration only fatal errors are logged now. You can change that in the debug section of the configuration file. You may also want to enable more verbose logging until you verified that the new Privoxy version is working as expected.
- Three other config file settings are now off by default: enable-remote-toggle, enable-remote-http-toggle, and enable-edit-actions. If you use or want these, you will need to explicitly enable them, and be aware of the security issues involved.
- The "filter-client-headers" and "filter-server-headers" actions that were introduced with Privoxy 3.0.5 to apply content filters to the headers have been removed and replaced with new actions. See the What's New section above.

4. Quickstart to Using Privoxy

- Install Privoxy. See the Installation Section below for platform specific information.
- Advanced users and those who want to offer Privoxy service to more than just their local machine should check the main config file, especially the security-relevant options. These are off by default.
- Start Privoxy, if the installation program has not done this already (may vary according to platform). See the section Starting Privoxy.
- Set your browser to use Privoxy as HTTP and HTTPS (SSL) proxy by setting the proxy configuration for address of `127.0.0.1` and port `8118`. *DO NOT* activate proxying for FTP or any protocols besides HTTP and HTTPS (SSL) unless you intend to prevent your browser from using these protocols.
- Flush your browser's disk and memory caches, to remove any cached ad images. If using Privoxy to manage cookies, you should remove any currently stored cookies too.
- A default installation should provide a reasonable starting point for most. There will undoubtedly be occasions where you will want to adjust the configuration, but that can be dealt with as the need arises. Little to no initial configuration is required in most cases, you may want to enable the web-based action editor though. Be sure to read the warnings first.

  See the Configuration section for more configuration options, and how to customize your installation. You might also want to look at the next section for a quick introduction to how Privoxy blocks ads and banners.
- If you experience ads that slip through, innocent images that are blocked, or otherwise feel the need to fine-tune Privoxy's behavior, take a look at the actions files. As a quick start, you might find the richly commented examples helpful. You can also view and edit the actions files through the web-based user interface. The Appendix "Troubleshooting: Anatomy of an Action" has hints on how to understand and debug actions that "misbehave".
- Please see the section Contacting the Developers on how to report bugs, problems with websites or to get help.
- Now enjoy surfing with enhanced control, comfort and privacy!

4.1. Quickstart to Ad Blocking

Ad blocking is but one of Privoxy's array of features. Many of these features are for the technically minded advanced user. But, ad and banner blocking is surely common ground for everybody.

This section will provide a quick summary of ad blocking so you can get up to speed quickly without having to read the more extensive information provided below, though this is highly recommended.

First a bit of a warning ... blocking ads is much like blocking SPAM: the more aggressive you are about it, the more likely you are to block things that were not intended. And the more likely that some things may not work as intended. So there is a trade off here. If you want extreme ad free browsing, be prepared to deal with more "problem" sites, and to spend more time adjusting the configuration to solve these unintended consequences. In short, there is not an easy way to eliminate *all* ads. Either take the easy way and settle for *most* ads blocked with the default configuration, or jump in and tweak it for your personal surfing habits and preferences.

Secondly, a brief explanation of Privoxy's "actions". "Actions" in this context, are the directives we use to tell Privoxy to perform some task relating to HTTP transactions (i.e. web browsing). We tell Privoxy to take some "action". Each action has a unique name and function. While there are many potential actions in Privoxy's arsenal, only a few are used for ad blocking. Actions, and action configuration files, are explained in depth below.

Actions are specified in Privoxy's configuration, followed by one or more URLs to which the action should apply. URLs can actually be URL type patterns that use wildcards so they can apply potentially to a range of similar URLs. The actions, together with the URL patterns are called a section.

When you connect to a website, the full URL will either match one or more of the sections as defined in Privoxy's configuration, or not. If so, then Privoxy will perform the respective actions. If not, then nothing special happens. Furthermore, web pages may contain embedded, secondary URLs that your web browser will use to load additional components of the page, as it parses the original page's HTML content. An ad image for instance, is just an URL embedded in the page somewhere. The image itself may be on the same server, or a server somewhere else on the Internet. Complex web pages will have many such embedded URLs. Privoxy can deal with each URL individually, so, for instance, the main page text is not touched, but images from such-and-such server are blocked.

The most important actions for basic ad blocking are: `block`, `handle-as-image`, `handle-as-empty-document`,and `set-image-blocker`:

- `block` - this is perhaps the single most used action, and is particularly important for ad blocking. This action stops any contact between your browser and any URL patterns that match this action's configuration. It can be used for blocking ads, but also anything that is determined to be unwanted. By itself, it simply stops any communication with the remote server and sends Privoxy's own built-in BLOCKED page instead to let you now what has happened (with some exceptions, see below).
- `handle-as-image` - tells Privoxy to treat this URL as an image. Privoxy's default configuration already does this for all common image types (e.g. GIF), but there are many situations where this is not so easy to determine. So we'll force it in these cases. This is particularly important for ad blocking, since only if we know that it's an image of some kind, can we replace it with an image of our choosing, instead of the Privoxy BLOCKED page (which would only result in a "broken image" icon). There are some limitations to this though. For instance, you can't just brute-force an image substitution for an entire HTML page in most situations.

- `handle-as-empty-document` - sends an empty document instead of Privoxy's normal BLOCKED HTML page. This is useful for file types that are neither HTML nor images, such as blocking JavaScript files.
- `set-image-blocker` - tells Privoxy what to display in place of an ad image that has hit a block rule. For this to come into play, the URL must match a `block` action somewhere in the configuration, *and*, it must also match an `handle-as-image` action.

  The configuration options on what to display instead of the ad are:

  pattern - a checkerboard pattern, so that an ad replacement is obvious. This is the default.

  blank - A very small empty GIF image is displayed. This is the so-called "invisible" configuration option.

  http://<URL> - A redirect to any image anywhere of the user's choosing (advanced usage).

Advanced users will eventually want to explore Privoxy `filters` as well. Filters are very different from `blocks`. A "block" blocks a site, page, or unwanted contented. Filters are a way of filtering or modifying what is actually on the page. An example filter usage: a text replacement of "no-no" for "nasty-word". That is a very simple example. This process can be used for ad blocking, but it is more in the realm of advanced usage and has some pitfalls to be wary off.

The quickest way to adjust any of these settings is with your browser through the special Privoxy editor at http://config.privoxy.org/show-status (shortcut: http://p.p/show-status). This is an internal page, and does not require Internet access.

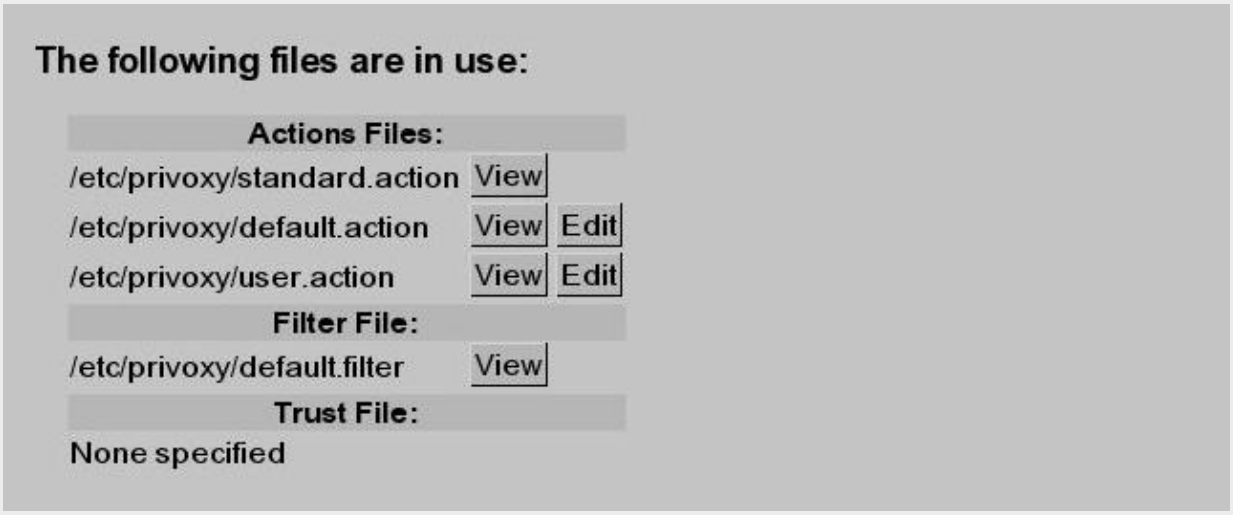
Note that as of Privoxy 3.0.7 beta the action editor is disabled by default. Check the enable-edit-actions section in the configuration file to learn why and in which cases it's safe to enable again.

If you decided to enable the action editor, select the appropriate "actions" file, and click "Edit". It is best to put personal or local preferences in `user.action` since this is not meant to be overwritten during upgrades, and will over-ride the settings in other files. Here you can insert new "actions", and URLs for ad blocking or other purposes, and make other adjustments to the configuration. Privoxy will detect these changes automatically.

A quick and simple step by step example:

- Right click on the ad image to be blocked, then select "Copy Link Location" from the pop-up menu.
- Set your browser to http://config.privoxy.org/show-status
- Find `user.action` in the top section, and click on "Edit":

Figure 1. Actions Files in Use



- You should have a section with only `block` listed under "Actions:". If not, click a "Insert new section below" button, and in the new section that just appeared, click the Edit button right under the word "Actions:". This will bring up a list of all actions. Find `block` near the top, and click in the "Enabled" column, then "Submit" just below the list.
- Now, in the `block` actions section, click the "Add" button, and paste the URL the browser got from "Copy Link Location". Remove the `http://` at the beginning of the URL. Then, click "Submit" (or "OK" if in a pop-up window).
- Now go back to the original page, and press **SHIFT-Reload** (or flush all browser caches). The image should be gone now.

This is a very crude and simple example. There might be good reasons to use a wildcard pattern match to include potentially similar images from the same site. For a more extensive explanation of "patterns", and the entire actions concept, see the Actions section.

For advanced users who want to hand edit their config files, you might want to now go to the Actions Files Tutorial. The ideas explained therein also apply to the web-based editor.

There are also various filters that can be used for ad blocking (filters are a special subset of actions). These fall into the "advanced" usage category, and are explained in depth in later sections.

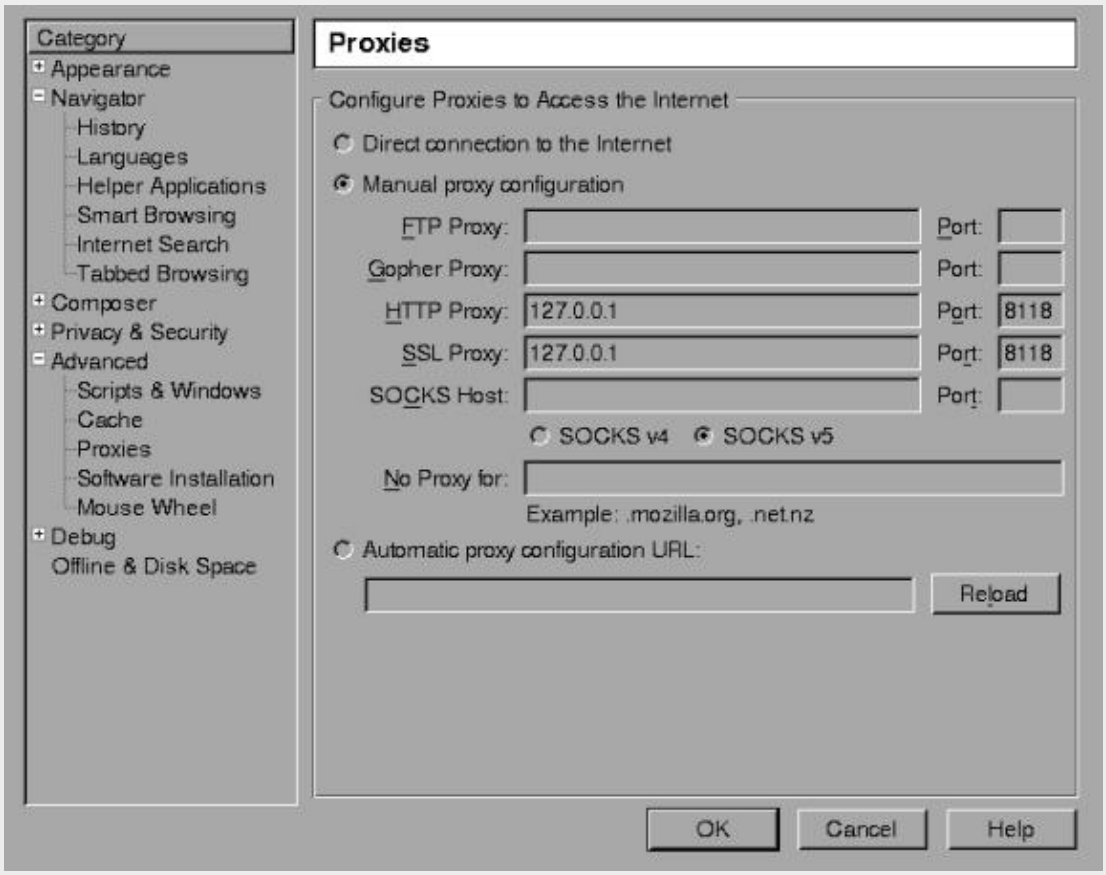
---

5. Starting Privoxy

Before launching Privoxy for the first time, you will want to configure your browser(s) to use Privoxy as a HTTP and HTTPS (SSL) proxy. The default is 127.0.0.1 (or localhost) for the proxy address, and port 8118 (earlier versions used port 8000). This is the one configuration step *that must be done*!

Please note that Privoxy can only proxy HTTP and HTTPS traffic. It will not work with FTP or other protocols.

Figure 2. Proxy Configuration Showing Mozilla/Netscape HTTP and HTTPS (SSL) Settings



With Firefox, this is typically set under:

```
Tools -> Options ->  Advanced -> Network ->Connection -> Settings
```

Or optionally on some platforms:

```
Edit -> Preferences -> General -> Connection Settings -> Manual Proxy Configuration
```

With Netscape (and Mozilla), this can be set under:

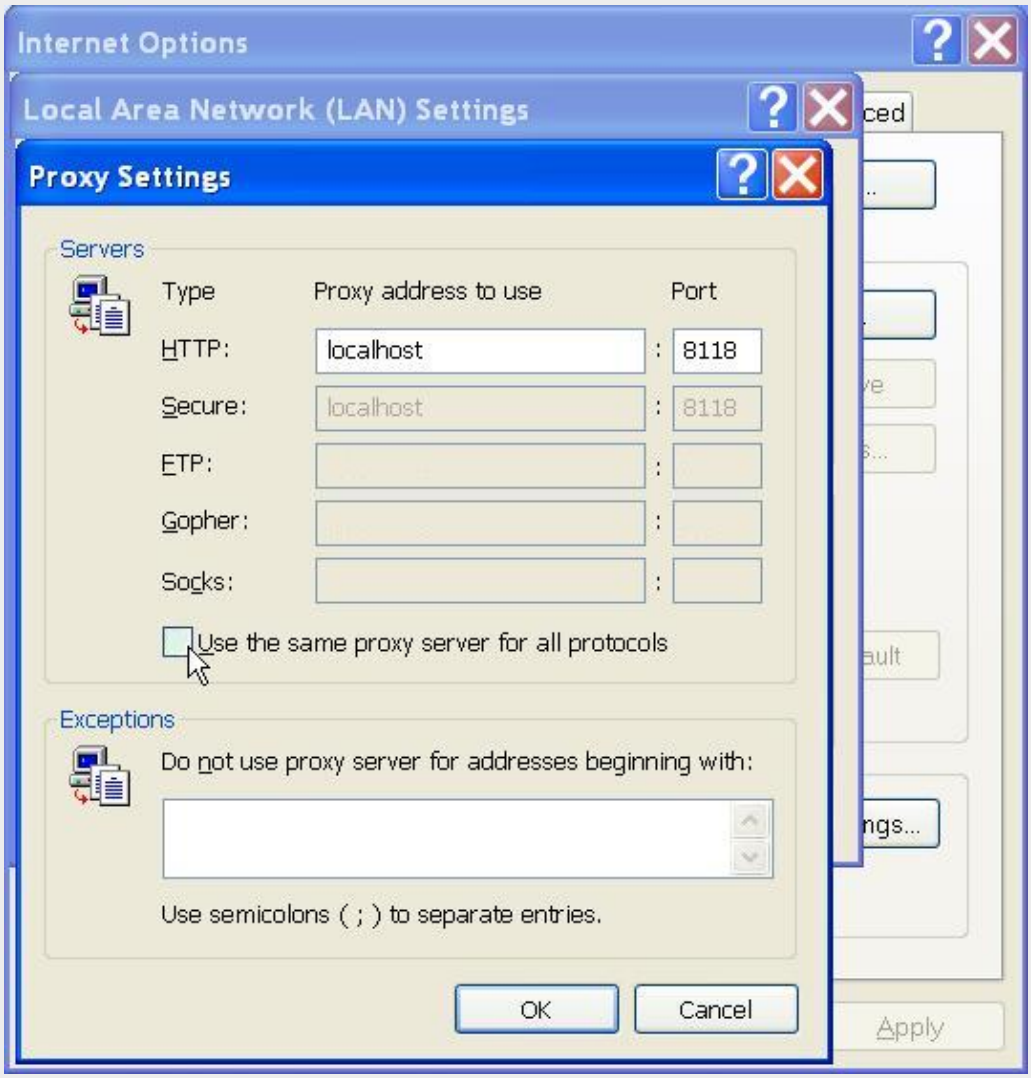
```
Edit -> Preferences -> Advanced -> Proxies -> HTTP Proxy
```

For Internet Explorer v.5-7:

```
Tools -> Internet Options -> Connections -> LAN Settings
```

Then, check "Use Proxy" and fill in the appropriate info (Address: 127.0.0.1, Port: 8118). Include HTTPS (SSL), if you want HTTPS proxy support too (sometimes labeled "Secure"). Make sure any checkboxes like "Use the same proxy server for all protocols" is *UNCHECKED*. You want only HTTP and HTTPS (SSL)!

Figure 3. Proxy Configuration Showing Internet Explorer HTTP and HTTPS (Secure) Settings



After doing this, flush your browser's disk and memory caches to force a re-reading of all pages and to get rid of any ads that may be cached. Remove any cookies, if you want Privoxy to manage that. You are now ready to start enjoying the benefits of using Privoxy!

Privoxy itself is typically started by specifying the main configuration file to be used on the command line. If no configuration file is specified on the command line, Privoxy will look for a file named `config` in the current directory. Except on Win32 where it will try `config.txt`.

5.1. Red Hat and Fedora

A default Red Hat installation may not start Privoxy upon boot. It will use the file `/etc/privoxy/config` as its main configuration file.

```
# /etc/rc.d/init.d/privoxy start
```

Or ...

```
# service privoxy start
```

5.2. Debian

We use a script. Note that Debian typically starts Privoxy upon booting per default. It will use the file `/etc/privoxy/config` as its main configuration file.

```
# /etc/init.d/privoxy start
```

5.3. Windows

Click on the Privoxy Icon to start Privoxy. If no configuration file is specified on the command line, Privoxy will look for a file named `config.txt`. Note that Windows will automatically start Privoxy when the system starts if you chose that option when installing.

Privoxy can run with full Windows service functionality. On Windows only, the Privoxy program has two new command line arguments to install and uninstall Privoxy as a service. See the Windows Installation instructions for details.

5.4. Solaris, NetBSD, FreeBSD, HP-UX and others

Example Unix startup command:

```
# /usr/sbin/privoxy /etc/privoxy/config
```

5.5. OS/2

During installation, Privoxy is configured to start automatically when the system restarts. You can start it manually by double-clicking on the Privoxy icon in the Privoxy folder.

5.6. Mac OSX

During installation, Privoxy is configured to start automatically when the system restarts. To start Privoxy manually, double-click on the `StartPrivoxy.command` icon in the `/Library/Privoxy` folder. Or, type this command in the Terminal:

```
 /Library/Privoxy/StartPrivoxy.command
```

You will be prompted for the administrator password.

5.7. AmigaOS

Start Privoxy (with RUN <>NIL:) in your `startnet` script (AmiTCP), in `s:user-startup` (RoadShow), as startup program in your startup script (Genesis), or as startup action (Miami and MiamiDx). Privoxy will automatically quit when you quit your TCP/IP stack (just ignore the harmless warning your TCP/IP stack may display that Privoxy is still running).

5.8. Gentoo

A script is again used. It will use the file `/etc/privoxy/config` as its main configuration file.

```
/etc/init.d/privoxy start
```

Note that Privoxy is not automatically started at boot time by default. You can change this with the `rc-update` command.

```
rc-update add privoxy default
```

5.9. Command Line Options

Privoxy may be invoked with the following command-line options:

- *--version*

  Print version info and exit. Unix only.
- *--help*

  Print short usage info and exit. Unix only.
- *--no-daemon*

  Don't become a daemon, i.e. don't fork and become process group leader, and don't detach from controlling tty. Unix only.
- *--pidfile FILE*

  On startup, write the process ID to *FILE*. Delete the *FILE* on exit. Failure to create or delete the *FILE* is non-fatal. If no *FILE* option is given, no PID file will be used. Unix only.
- *--user USER[.GROUP]*

  After (optionally) writing the PID file, assume the user ID of *USER*, and if included the GID of GROUP. Exit if the privileges are not sufficient to do so. Unix only.

- *--chroot*

  Before changing to the user ID given in the *--user* option, chroot to that user's home directory, i.e. make the kernel pretend to the Privoxy process that the directory tree starts there. If set up carefully, this can limit the impact of possible vulnerabilities in Privoxy to the files contained in that hierarchy. Unix only.
- *--pre-chroot-nslookup hostname*

  Specifies a hostname to look up before doing a chroot. On some systems, initializing the resolver library involves reading config files from /etc and/or loading additional shared libraries from /lib. On these systems, doing a hostname lookup before the chroot reduces the number of files that must be copied into the chroot tree.

  For fastest startup speed, a good value is a hostname that is not in /etc/hosts but that your local name server (listed in /etc/resolv.conf) can resolve without recursion (that is, without having to ask any other name servers). The hostname need not exist, but if it doesn't, an error message (which can be ignored) will be output.
- *configfile*

  If no *configfile* is included on the command line, Privoxy will look for a file named "config" in the current directory (except on Win32 where it will look for "config.txt" instead). Specify full path to avoid confusion. If no config file is found, Privoxy will fail to start.

On MS Windows only there are two additional command-line options to allow Privoxy to install and run as a *service*. See the Window Installation section for details.

---

6. Privoxy Configuration

All Privoxy configuration is stored in text files. These files can be edited with a text editor. Many important aspects of Privoxy can also be controlled easily with a web browser.

6.1. Controlling Privoxy with Your Web Browser

Privoxy's user interface can be reached through the special URL http://config.privoxy.org/ (shortcut: http://p.p/), which is a built-in page and works without Internet access. You will see the following section:

Privoxy Menu

▪ View & change the current configuration
▪ View the source code version numbers
▪ View the request headers.
▪ Look up which actions apply to a URL and why
▪ Toggle Privoxy on or off
▪ Documentation

This should be self-explanatory. Note the first item leads to an editor for the actions files, which is where the ad, banner, cookie, and URL blocking magic is configured as well as other advanced features of Privoxy. This is an easy way to adjust various aspects of Privoxy configuration. The actions file, and other configuration files, are explained in detail below.

"Toggle Privoxy On or Off" is handy for sites that might have problems with your current actions and filters. You can in fact use it as a test to see whether it is Privoxy causing the problem or not. Privoxy continues to run as a proxy in this case, but all manipulation is disabled, i.e. Privoxy acts like a normal forwarding proxy. There is even a toggle Bookmarklet offered, so that you can toggle Privoxy with one click from your browser.

Note that several of the features described above are disabled by default in Privoxy 3.0.7 beta and later. Check the configuration file to learn why and in which cases it's safe to enable them again.

6.2. Configuration Files Overview

For Unix, *BSD and Linux, all configuration files are located in `/etc/privoxy/` by default. For MS Windows, OS/2, and AmigaOS these are all in the same directory as the Privoxy executable.

The installed defaults provide a reasonable starting point, though some settings may be aggressive by some standards. For the time being, the principle configuration files are:

- The main configuration file is named `config` on Linux, Unix, BSD, OS/2, and AmigaOS and `config.txt` on Windows. This is a required file.
- `default.action` (the main actions file) is used to define which "actions" relating to banner-blocking, images, pop-ups, content modification, cookie handling etc should be applied by default. It also defines many exceptions (both positive and negative) from this default set of actions that enable Privoxy to selectively eliminate the junk, and only the junk, on as many websites as possible.

  Multiple actions files may be defined in `config`. These are processed in the order they are defined. Local customizations and locally preferred exceptions to the default policies as defined in `default.action` (which you will most probably want to define sooner or later) are probably best applied in `user.action`, where you can preserve them across upgrades. `standard.action` is only for Privoxy's internal use.

  There is also a web based editor that can be accessed from http://config.privoxy.org/show-status (Shortcut: http://p.p/show-status) for the various actions files.
- "Filter files" (the filter file) can be used to re-write the raw page content, including viewable text as well as embedded HTML and JavaScript, and whatever else lurks on any given web page. The filtering jobs are only pre-defined here; whether to apply them or not is up to the actions files. `default.filter` includes various filters made available for use by the developers. Some are much more intrusive than others, and all should be used with caution. You may define additional filter files in `config` as you can with actions files. We suggest `user.filter` for any locally defined filters or customizations.

The syntax of the configuration and filter files may change between different Privoxy versions, unfortunately some enhancements cost backwards compatibility.

All files use the "#" character to denote a comment (the rest of the line will be ignored) and understand line continuation through placing a backslash ("\") as the very last character in a line. If the # is preceded by a backslash, it looses its special function. Placing a # in front of an otherwise valid configuration line to prevent it from being interpreted is called "commenting out" that line. Blank lines are ignored.

The actions files and filter files can use Perl style regular expressions for maximum flexibility.

After making any changes, there is no need to restart Privoxy in order for the changes to take effect. Privoxy detects such changes automatically. Note, however, that it may take one or two additional requests for the change to take effect. When changing the listening address of Privoxy, these "wake up" requests must obviously be sent to the *old* listening address.

---

7. The Main Configuration File

Again, the main configuration file is named `config` on Linux/Unix/BSD and OS/2, and `config.txt` on Windows. Configuration lines consist of an initial keyword followed by a list of values, all separated by whitespace (any number of spaces or tabs). For example:

```
confdir /etc/privoxy
```

Assigns the value `/etc/privoxy` to the option `confdir` and thus indicates that the configuration directory is named "/etc/privoxy/".

All options in the config file except for `confdir` and `logdir` are optional. Watch out in the below description for what happens if you leave them unset.

The main config file controls all aspects of Privoxy's operation that are not location dependent (i.e. they apply universally, no matter where you may be surfing).

7.1. Local Set-up Documentation

If you intend to operate Privoxy for more users than just yourself, it might be a good idea to let them know how to reach you, what you block and why you do that, your policies, etc.

7.1.1. user-manual

Specifies:
: Location of the Privoxy User Manual.

Type of value:
: A fully qualified URI

Default value:
: *Unset*

Effect if unset:
: http://www.privoxy.org/*version*/user-manual/ will be used, where *version* is the Privoxy version.

Notes:
: The User Manual URI is the single best source of information on Privoxy, and is used for help links from some of the internal CGI pages. The manual itself is normally packaged with the binary distributions, so you probably want to set this to a locally installed copy.

  Examples:

  The best all purpose solution is simply to put the full local PATH to where the *User Manual* is located:

  ```
  user-manual  /usr/share/doc/privoxy/user-manual
  ```

  The User Manual is then available to anyone with access to Privoxy, by following the built-in URL: `http://config.privoxy.org/user-manual/` (or the shortcut: `http://p.p/user-manual/`).

  If the documentation is not on the local system, it can be accessed from a remote server, as:

  ```
  user-manual  http://example.com/privoxy/user-manual/
  ```

Warning
If set, this option should be *the first option in the config file*, because it is used while the config file is being read on start-up.

7.1.2. trust-info-url

Specifies:
: A URL to be displayed in the error page that users will see if access to an untrusted page is denied.

Type of value:
: URL

Default value:
: Two example URLs are provided

Effect if unset:
: No links are displayed on the "untrusted" error page.

Notes:
: The value of this option only matters if the experimental trust mechanism has been activated. (See *trustfile* below.)

  If you use the trust mechanism, it is a good idea to write up some on-line documentation about your trust policy and to specify the URL(s) here. Use multiple times for multiple URLs.

The URL(s) should be added to the trustfile as well, so users don't end up locked out from the information on why they were locked out in the first place!

7.1.3. admin-address

Specifies:
: An email address to reach the Privoxy administrator.

Type of value:
: Email address

Default value:
: *Unset*

Effect if unset:
: No email address is displayed on error pages and the CGI user interface.

Notes:
: If both `admin-address` and `proxy-info-url` are unset, the whole "Local Privoxy Support" box on all generated pages will not be shown.

7.1.4. proxy-info-url

Specifies:
: A URL to documentation about the local Privoxy setup, configuration or policies.

Type of value:
: URL

Default value:
: *Unset*

Effect if unset:
: No link to local documentation is displayed on error pages and the CGI user interface.

Notes:
: If both `admin-address` and `proxy-info-url` are unset, the whole "Local Privoxy Support" box on all generated pages will not be shown.

  This URL shouldn't be blocked ;-)

7.2. Configuration and Log File Locations

Privoxy can (and normally does) use a number of other files for additional configuration, help and logging. This section of the configuration file tells Privoxy where to find those other files.

The user running Privoxy, must have read permission for all configuration files, and write permission to any files that would be modified, such as log files and actions files.

7.2.1. confdir

Specifies:
: The directory where the other configuration files are located.

Type of value:
: Path name

Default value:
: /etc/privoxy (Unix) *or* Privoxy installation dir (Windows)

Effect if unset:
: *Mandatory*

Notes:
: No trailing "`/`", please.

7.2.2. templdir

Specifies:
: An alternative directory where the templates are loaded from.

Type of value:
: Path name

Default value:
: unset

Effect if unset:
: The templates are assumed to be located in confdir/template.

Notes:
: Privoxy's original templates are usually overwritten with each update. Use this option to relocate customized templates that should be kept. As template variables might change between updates, you shouldn't expect templates to work with Privoxy releases other than the one they were part of, though.

7.2.3. logdir

Specifies:
The directory where all logging takes place (i.e. where `logfile` and `jarfile` are located).
Type of value:
Path name
Default value:
/var/log/privoxy (Unix) *or* Privoxy installation dir (Windows)
Effect if unset:
Mandatory
Notes:
No trailing "`/`", please.

7.2.4. actionsfile

Specifies:
The actions file(s) to use
Type of value:
Complete file name, relative to `confdir`
Default values:

```
standard.action     # Internal purposes, no editing recommended
default.action      # Main actions file
user.action         # User customizations
```

Effect if unset:
No actions are taken at all. More or less neutral proxying.
Notes:
Multiple `actionsfile` lines are permitted, and are in fact recommended!

The default values include `standard.action`, which is used for internal purposes and should be loaded, `default.action`, which is the "main" actions file maintained by the developers, and `user.action`, where you can make your personal additions.

Actions files contain all the per site and per URL configuration for ad blocking, cookie management, privacy considerations, etc. There is no point in using Privoxy without at least one actions file.

Note that since Privoxy 3.0.7, the complete filename, including the ".action" extension has to be specified. The syntax change was necessary to be consistent with the other file options and to allow previously forbidden characters.

7.2.5. filterfile

Specifies:
The filter file(s) to use
Type of value:
File name, relative to `confdir`
Default value:
default.filter (Unix) *or* default.filter.txt (Windows)
Effect if unset:
No textual content filtering takes place, i.e. all +`filter`{*name*} actions in the actions files are turned neutral.
Notes:
Multiple `filterfile` lines are permitted.

The filter files contain content modification rules that use regular expressions. These rules permit powerful changes on the content of Web pages, and optionally the headers as well, e.g., you could try to disable your favorite JavaScript annoyances, re-write the actual displayed text, or just have some fun playing buzzword bingo with web pages.

The +`filter`{*name*} actions rely on the relevant filter (*name*) to be defined in a filter file!

A pre-defined filter file called `default.filter` that contains a number of useful filters for common problems is included in the distribution. See the section on the `filter` action for a list.

It is recommended to place any locally adapted filters into a separate file, such as `user.filter`.

7.2.6. logfile

Specifies:
The log file to use
Type of value:
File name, relative to `logdir`
Default value:
Unset (commented out). When activated: logfile (Unix) *or* privoxy.log (Windows).
Effect if unset:
No logfile is written.
Notes:
The logfile is where all logging and error messages are written. The level of detail and number of messages are set with the `debug` option (see below). The logfile can be useful for tracking down a problem with Privoxy (e.g., it's not blocking an ad you think it should block) and it can help you to monitor what your browser is doing.

Depending on the debug options below, the logfile may be a privacy risk if third parties can get access to it. As most users will never look at it, Privoxy 3.0.7 and later only log fatal errors by default.

For most troubleshooting purposes, you will have to change that, please refer to the debugging section for details.

Your logfile will grow indefinitely, and you will probably want to periodically remove it. On Unix systems, you can do this with a cron job (see "man cron"). For Red Hat based Linux distributions, a **logrotate** script has been included.

Any log files must be writable by whatever user Privoxy is being run as (on Unix, default user id is "privoxy").

7.2.7. jarfile

Specifies:
The file to store intercepted cookies in
Type of value:
File name, relative to `logdir`
Default value:
Unset (commented out). When activated: jarfile (Unix) *or* privoxy.jar (Windows).
Effect if unset:
Intercepted cookies are not stored in a dedicated log file.
Notes:
The jarfile may grow to ridiculous sizes over time.

If debug 8 (show header parsing) is enabled, cookies are also written to the logfile with the rest of the headers. Therefore this option isn't very useful and may be removed in future releases. Please report to the developers if you are still using it.

7.2.8. trustfile

Specifies:
The name of the trust file to use
Type of value:
File name, relative to `confdir`
Default value:
Unset (commented out). When activated: trust (Unix) *or* trust.txt (Windows)
Effect if unset:
The entire trust mechanism is disabled.
Notes:
The trust mechanism is an experimental feature for building white-lists and should be used with care. It is *NOT* recommended for the casual user.

If you specify a trust file, Privoxy will only allow access to sites that are specified in the trustfile. Sites can be listed in one of two ways:

Prepending a ~ character limits access to this site only (and any sub-paths within this site), e.g. `~www.example.com` allows access to `~www.example.com/features/news.html`, etc.

Or, you can designate sites as *trusted referrers*, by prepending the name with a + character. The effect is that access to untrusted sites will be granted -- but only if a link from this trusted referrer was used to get there. The link target will then be added to the "trustfile" so that future, direct accesses will be granted. Sites added via this mechanism do not become trusted referrers themselves (i.e. they are added with a ~ designation). There is a limit of 512 such entries, after which new entries will not be made.

If you use the + operator in the trust file, it may grow considerably over time.

It is recommended that Privoxy be compiled with the `--disable-force`, `--disable-toggle` and `--disable-editor` options, if this feature is to be used.

Possible applications include limiting Internet access for children.

7.3. Debugging

These options are mainly useful when tracing a problem. Note that you might also want to invoke Privoxy with the `--no-daemon` command line option when debugging.

7.3.1. debug

Specifies:
Key values that determine what information gets logged.
Type of value:
Integer values
Default value:
0 (i.e.: only fatal errors (that cause Privoxy to exit) are logged)
Effect if unset:
Default value is used (see above).
Notes:
The available debug levels are:

```
debug         1 # log each request destination (and the crunch reason if Privoxy intercepted the request)
debug         2 # show each connection status
debug         4 # show I/O status
debug         8 # show header parsing
debug        16 # log all data written to the network into the logfile
debug        32 # debug force feature
debug        64 # debug regular expression filters
debug       128 # debug redirects
debug       256 # debug GIF de-animation
debug       512 # Common Log Format
debug      1024 # debug kill pop-ups
debug      2048 # CGI user interface
debug      4096 # Startup banner and warnings.
debug      8192 # Non-fatal errors
```

To select multiple debug levels, you can either add them or use multiple `debug` lines.

A debug level of 1 is informative because it will show you each request as it happens. *1, 4096 and 8192 are recommended* so that you will notice when things go wrong. The other levels are probably only of interest if you are hunting down a specific problem. They can produce a hell of an output (especially 16).

Privoxy used to ship with the debug levels recommended above enabled by default, but due to privacy concerns 3.0.7 and later are configured to only log fatal errors.

If you are used to the more verbose settings, simply enable the debug lines below again.

If you want to use pure CLF (Common Log Format), you should set "debug 512" *ONLY* and not enable anything else.

Privoxy has a hard-coded limit for the length of log messages. If it's reached, messages are logged truncated and marked with "... [too long, truncated]".

Please don't file any support requests without trying to reproduce the problem with increased debug level first. Once you read the log messages, you may even be able to solve the problem on your own.

7.3.2. single-threaded

Specifies:
Whether to run only one server thread.
Type of value:
None
Default value:
Unset
Effect if unset:
Multi-threaded (or, where unavailable: forked) operation, i.e. the ability to serve multiple requests simultaneously.
Notes:
This option is only there for debugging purposes. *It will drastically reduce performance.*

7.4. Access Control and Security

This section of the config file controls the security-relevant aspects of Privoxy's configuration.

7.4.1. listen-address

Specifies:
: The IP address and TCP port on which Privoxy will listen for client requests.

Type of value:
: [*IP-Address*]:*Port*

Default value:
: 127.0.0.1:8118

Effect if unset:
: Bind to 127.0.0.1 (localhost), port 8118. This is suitable and recommended for home users who run Privoxy on the same machine as their browser.

Notes:
: You will need to configure your browser(s) to this proxy address and port.

  If you already have another service running on port 8118, or if you want to serve requests from other machines (e.g. on your local network) as well, you will need to override the default.

  If you leave out the IP address, Privoxy will bind to all interfaces (addresses) on your machine and may become reachable from the Internet. In that case, consider using access control lists (ACL's, see below), and/or a firewall.

  If you open Privoxy to untrusted users, you will also want to make sure that the following actions are disabled: `enable-edit-actions` and `enable-remote-toggle`

Example:
: Suppose you are running Privoxy on a machine which has the address 192.168.0.1 on your local private network (192.168.0.0) and has another outside connection with a different address. You want it to serve requests from inside only:

```
listen-address  192.168.0.1:8118
```

7.4.2. toggle

Specifies:
: Initial state of "toggle" status

Type of value:
: 1 or 0

Default value:
: 1

Effect if unset:
: Act as if toggled on

Notes:
: If set to 0, Privoxy will start in "toggled off" mode, i.e. mostly behave like a normal, content-neutral proxy with both ad blocking and content filtering disabled. See `enable-remote-toggle` below.

  The windows version will only display the toggle icon in the system tray if this option is present.

7.4.3. enable-remote-toggle

Specifies:
: Whether or not the web-based toggle feature may be used

Type of value:
: 0 or 1

Default value:
: 0

Effect if unset:
: The web-based toggle feature is disabled.

Notes:
: When toggled off, Privoxy mostly acts like a normal, content-neutral proxy, i.e. doesn't block ads or filter content.

  Access to the toggle feature can *not* be controlled separately by "ACLs" or HTTP authentication, so that everybody who can access Privoxy (see "ACLs" and `listen-address` above) can toggle it for all users. So this option is *not recommended* for multi-user environments with untrusted users.

  Note that malicious client side code (e.g Java) is also capable of using this option.

  As a lot of Privoxy users don't read documentation, this feature is disabled by default.

Note that you must have compiled Privoxy with support for this feature, otherwise this option has no effect.

7.4.4. enable-remote-http-toggle

Specifies:
Whether or not Privoxy recognizes special HTTP headers to change its behaviour.
Type of value:
0 or 1
Default value:
0
Effect if unset:
Privoxy ignores special HTTP headers.
Notes:
When toggled on, the client can change Privoxy's behaviour by setting special HTTP headers. Currently the only supported special header is "X-Filter: No", to disable filtering for the ongoing request, even if it is enabled in one of the action files.

This feature is disabled by default. If you are using Privoxy in a environment with trusted clients, you may enable this feature at your discretion. Note that malicious client side code (e.g Java) is also capable of using this feature.

This option will be removed in future releases as it has been obsoleted by the more general header taggers.

7.4.5. enable-edit-actions

Specifies:
Whether or not the web-based actions file editor may be used
Type of value:
0 or 1
Default value:
0
Effect if unset:
The web-based actions file editor is disabled.
Notes:
Access to the editor can *not* be controlled separately by "ACLs" or HTTP authentication, so that everybody who can access Privoxy (see "ACLs" and `listen-address` above) can modify its configuration for all users.

This option is *not recommended* for environments with untrusted users and as a lot of Privoxy users don't read documentation, this feature is disabled by default.

Note that malicious client side code (e.g Java) is also capable of using the actions editor and you shouldn't enable this options unless you understand the consequences and are sure your browser is configured correctly.

Note that you must have compiled Privoxy with support for this feature, otherwise this option has no effect.

7.4.6. enforce-blocks

Specifies:
Whether the user is allowed to ignore blocks and can "go there anyway".
Type of value:
`0 or 1`
Default value:
`0`
Effect if unset:
Blocks are not enforced.
Notes:
Privoxy is mainly used to block and filter requests as a service to the user, for example to block ads and other junk that clogs the pipes. Privoxy's configuration isn't perfect and sometimes innocent pages are blocked. In this situation it makes sense to allow the user to enforce the request and have Privoxy ignore the block.

In the default configuration Privoxy's "Blocked" page contains a "go there anyway" link to adds a special string (the force prefix) to the request URL. If that link is used, Privoxy will detect the force prefix, remove it again and let the request pass.

Of course Privoxy can also be used to enforce a network policy. In that case the user obviously should not be able to bypass any blocks, and that's what the "enforce-blocks" option is for. If it's enabled, Privoxy hides the "go there anyway" link. If the user adds the force prefix by hand, it will not be accepted and the circumvention attempt is logged.
Examples:
enforce-blocks 1

7.4.7. ACLs: permit-access and deny-access

Specifies:
: Who can access what.

Type of value:
: *src_addr*[/*src_masklen*] [*dst_addr*[/*dst_masklen*]]

  Where *src_addr* and *dst_addr* are IP addresses in dotted decimal notation or valid DNS names, and *src_masklen* and *dst_masklen* are subnet masks in CIDR notation, i.e. integer values from 2 to 30 representing the length (in bits) of the network address. The masks and the whole destination part are optional.

Default value:
: *Unset*

Effect if unset:
: Don't restrict access further than implied by `listen-address`

Notes:
: Access controls are included at the request of ISPs and systems administrators, and *are not usually needed by individual users*. For a typical home user, it will normally suffice to ensure that Privoxy only listens on the localhost (127.0.0.1) or internal (home) network address by means of the *listen-address* option.

  Please see the warnings in the FAQ that Privoxy is not intended to be a substitute for a firewall or to encourage anyone to defer addressing basic security weaknesses.

  Multiple ACL lines are OK. If any ACLs are specified, Privoxy only talks to IP addresses that match at least one `permit-access` line and don't match any subsequent `deny-access` line. In other words, the last match wins, with the default being `deny-access`.

  If Privoxy is using a forwarder (see `forward` below) for a particular destination URL, the *dst_addr* that is examined is the address of the forwarder and *NOT* the address of the ultimate target. This is necessary because it may be impossible for the local Privoxy to determine the IP address of the ultimate target (that's often what gateways are used for).

  You should prefer using IP addresses over DNS names, because the address lookups take time. All DNS names must resolve! You can *not* use domain patterns like "*.org" or partial domain names. If a DNS name resolves to multiple IP addresses, only the first one is used.

  Denying access to particular sites by ACL may have undesired side effects if the site in question is hosted on a machine which also hosts other sites (most sites are).

Examples:
: Explicitly define the default behavior if no ACL and `listen-address` are set: "localhost" is OK. The absence of a *dst_addr* implies that *all* destination addresses are OK:

```
permit-access  localhost
```

  Allow any host on the same class C subnet as www.privoxy.org access to nothing but www.example.com (or other domains hosted on the same system):

```
permit-access  www.privoxy.org/24 www.example.com/32
```

  Allow access from any host on the 26-bit subnet 192.168.45.64 to anywhere, with the exception that 192.168.45.73 may not access the IP address behind www.dirty-stuff.example.com:

```
permit-access  192.168.45.64/26
deny-access    192.168.45.73    www.dirty-stuff.example.com
```

7.4.8. buffer-limit

Specifies:
: Maximum size of the buffer for content filtering.

Type of value:
: Size in Kbytes

Default value:
: 4096

Effect if unset:
: Use a 4MB (4096 KB) limit.

Notes:
: For content filtering, i.e. the `+filter` and `+deanimate-gif` actions, it is necessary that Privoxy buffers the entire document body. This can be potentially dangerous, since a server could just keep sending data indefinitely and wait for your RAM to exhaust -- with nasty consequences. Hence this option.

  When a document buffer size reaches the `buffer-limit`, it is flushed to the client unfiltered and no further attempt to filter the rest of the document is made. Remember that there may be multiple threads running, which might require up to

`buffer-limit` Kbytes *each*, unless you have enabled "single-threaded" above.

7.5. Forwarding

This feature allows routing of HTTP requests through a chain of multiple proxies.

Forwarding can be used to chain Privoxy with a caching proxy to speed up browsing. Using a parent proxy may also be necessary if the machine that Privoxy runs on has no direct Internet access.

Note that parent proxies can severely decrease your privacy level. For example a parent proxy could add your IP address to the request headers and if it's a caching proxy it may add the "Etag" header to revalidation requests again, even though you configured Privoxy to remove it. It may also ignore Privoxy's header time randomization and use the original values which could be used by the server as cookie replacement to track your steps between visits.

Also specified here are SOCKS proxies. Privoxy supports the SOCKS 4 and SOCKS 4A protocols.

7.5.1. forward

Specifies:
To which parent HTTP proxy specific requests should be routed.
Type of value:
`target_pattern` `http_parent`[:*`port`*]

where *`target_pattern`* is a URL pattern that specifies to which requests (i.e. URLs) this forward rule shall apply. Use `/` to denote "all URLs". `http_parent`[:*`port`*] is the DNS name or IP address of the parent HTTP proxy through which the requests should be forwarded, optionally followed by its listening port (default: 8080). Use a single dot (`.`) to denote "no forwarding".
Default value:
Unset
Effect if unset:
Don't use parent HTTP proxies.
Notes:
If *`http_parent`* is ".", then requests are not forwarded to another HTTP proxy but are made directly to the web servers.

Multiple lines are OK, they are checked in sequence, and the last match wins.
Examples:
Everything goes to an example parent proxy, except SSL on port 443 (which it doesn't handle):

```
forward   /      parent-proxy.example.org:8080
forward   :443   .
```

Everything goes to our example ISP's caching proxy, except for requests to that ISP's sites:

```
forward   /                 caching-proxy.isp.example.net:8000
forward   .isp.example.net  .
```

7.5.2. forward-socks4 and forward-socks4a

Specifies:
Through which SOCKS proxy (and optionally to which parent HTTP proxy) specific requests should be routed.
Type of value:
`target_pattern` `socks_proxy`[:*`port`*] `http_parent`[:*`port`*]

where *`target_pattern`* is a URL pattern that specifies to which requests (i.e. URLs) this forward rule shall apply. Use `/` to denote "all URLs". `http_parent` and *`socks_proxy`* are IP addresses in dotted decimal notation or valid DNS names (*`http_parent`* may be "." to denote "no HTTP forwarding"), and the optional *`port`* parameters are TCP ports, i.e. integer values from 1 to 64535
Default value:
Unset
Effect if unset:
Don't use SOCKS proxies.
Notes:
Multiple lines are OK, they are checked in sequence, and the last match wins.

The difference between `forward-socks4` and `forward-socks4a` is that in the SOCKS 4A protocol, the DNS resolution of the target hostname happens on the SOCKS server, while in SOCKS 4 it happens locally.

If *`http_parent`* is ".", then requests are not forwarded to another HTTP proxy but are made (HTTP-wise) directly to the web servers, albeit through a SOCKS proxy.
Examples:

From the company example.com, direct connections are made to all "internal" domains, but everything outbound goes through their ISP's proxy by way of example.com's corporate SOCKS 4A gateway to the Internet.

```
forward-socks4a   /              socks-gw.example.com:1080  www-cache.isp.example.net:8080
forward           .example.com   .
```

A rule that uses a SOCKS 4 gateway for all destinations but no HTTP parent looks like this:

```
forward-socks4   /              socks-gw.example.com:1080  .
```

To chain Privoxy and Tor, both running on the same system, you would use something like:

```
forward-socks4a   /               127.0.0.1:9050 .
```

The public Tor network can't be used to reach your local network, if you need to access local servers you therefore might want to make some exceptions:

```
forward         192.168.*.*/     .
forward            10.*.*.*/     .
forward           127.*.*.*/     .
```

Unencrypted connections to systems in these address ranges will be as (un)secure as the local network is, but the alternative is that you can't reach the local network through Privoxy at all. Of course this may actually be desired and there is no reason to make these exceptions if you aren't sure you need them.

If you also want to be able to reach servers in your local network by using their names, you will need additional exceptions that look like this:

```
forward           localhost/     .
```

7.5.3. Advanced Forwarding Examples

If you have links to multiple ISPs that provide various special content only to their subscribers, you can configure multiple Privoxies which have connections to the respective ISPs to act as forwarders to each other, so that *your* users can see the internal content of all ISPs.

Assume that host-a has a PPP connection to isp-a.example.net. And host-b has a PPP connection to isp-b.example.org. Both run Privoxy. Their forwarding configuration can look like this:

host-a:

```
forward    /           .
forward    .isp-b.example.net  host-b:8118
```

host-b:

```
forward    /           .
forward    .isp-a.example.org  host-a:8118
```

Now, your users can set their browser's proxy to use either host-a or host-b and be able to browse the internal content of both isp-a and isp-b.

If you intend to chain Privoxy and squid locally, then chaining as `browser -> squid -> privoxy` is the recommended way.

Assuming that Privoxy and squid run on the same box, your squid configuration could then look like this:

```
# Define Privoxy as parent proxy (without ICP)
cache_peer 127.0.0.1 parent 8118 7 no-query

# Define ACL for protocol FTP
acl ftp proto FTP

# Do not forward FTP requests to Privoxy
always_direct allow ftp

# Forward all the rest to Privoxy
never_direct allow all
```

You would then need to change your browser's proxy settings to squid's address and port. Squid normally uses port 3128. If unsure consult `http_port` in `squid.conf`.

You could just as well decide to only forward requests you suspect of leading to Windows executables through a virus-scanning parent proxy, say, on `antivir.example.com`, port 8010:

```
forward   /                          .
forward   /.*\.(exe|com|dll|zip)$    antivir.example.com:8010
```

7.5.4. forwarded-connect-retries

Specifies:
How often Privoxy retries if a forwarded connection request fails.
Type of value:
Number of retries.
Default value:
0
Effect if unset:
Connections forwarded through other proxies are treated like direct connections and no retry attempts are made.
Notes:
forwarded-connect-retries is mainly interesting for socks4a connections, where Privoxy can't detect why the connections failed. The connection might have failed because of a DNS timeout in which case a retry makes sense, but it might also have failed because the server doesn't exist or isn't reachable. In this case the retry will just delay the appearance of Privoxy's error message.

Note that in the context of this option, "forwarded connections" includes all connections that Privoxy forwards through other proxies. This option is not limited to the HTTP CONNECT method.

Only use this option, if you are getting lots of forwarding-related error messages that go away when you try again manually. Start with a small value and check Privoxy's logfile from time to time, to see how many retries are usually needed.
Examples:
forwarded-connect-retries 1

7.5.5. accept-intercepted-requests

Specifies:
Whether intercepted requests should be treated as valid.
Type of value:
0 or 1
Default value:
0
Effect if unset:
Only proxy requests are accepted, intercepted requests are treated as invalid.
Notes:
If you don't trust your clients and want to force them to use Privoxy, enable this option and configure your packet filter to redirect outgoing HTTP connections into Privoxy.

Make sure that Privoxy's own requests aren't redirected as well. Additionally take care that Privoxy can't intentionally connect to itself, otherwise you could run into redirection loops if Privoxy's listening port is reachable by the outside or an attacker has access to the pages you visit.
Examples:
accept-intercepted-requests 1

7.5.6. allow-cgi-request-crunching

Specifies:
Whether requests to Privoxy's CGI pages can be blocked or redirected.
Type of value:
0 or 1
Default value:
0
Effect if unset:
Privoxy ignores block and redirect actions for its CGI pages.
Notes:
By default Privoxy ignores block or redirect actions for its CGI pages. Intercepting these requests can be useful in multi-user setups to implement fine-grained access control, but it can also render the complete web interface useless and make debugging problems painful if done without care.

Don't enable this option unless you're sure that you really need it.
Examples:
allow-cgi-request-crunching 1

7.5.7. split-large-forms

Specifies:
Whether the CGI interface should stay compatible with broken HTTP clients.
Type of value:
0 or 1

Default value:
0
Effect if unset:
The CGI form generate long GET URLs.
Notes:
Privoxy's CGI forms can lead to rather long URLs. This isn't a problem as far as the HTTP standard is concerned, but it can confuse clients with arbitrary URL length limitations.

Enabling split-large-forms causes Privoxy to divide big forms into smaller ones to keep the URL length down. It makes editing a lot less convenient and you can no longer submit all changes at once, but at least it works around this browser bug.

If you don't notice any editing problems, there is no reason to enable this option, but if one of the submit buttons appears to be broken, you should give it a try.
Examples:
split-large-forms 1

7.6. Windows GUI Options

Privoxy has a number of options specific to the Windows GUI interface:

If "activity-animation" is set to 1, the Privoxy icon will animate when "Privoxy" is active. To turn off, set to 0.

```
activity-animation 1
```

If "log-messages" is set to 1, Privoxy will log messages to the console window:

```
log-messages 1
```

If "log-buffer-size" is set to 1, the size of the log buffer, i.e. the amount of memory used for the log messages displayed in the console window, will be limited to "log-max-lines" (see below).

Warning: Setting this to 0 will result in the buffer to grow infinitely and eat up all your memory!

```
log-buffer-size 1
```

log-max-lines is the maximum number of lines held in the log buffer. See above.

```
log-max-lines 200
```

If "log-highlight-messages" is set to 1, Privoxy will highlight portions of the log messages with a bold-faced font:

```
log-highlight-messages 1
```

The font used in the console window:

```
log-font-name Comic Sans MS
```

Font size used in the console window:

```
log-font-size 8
```

"show-on-task-bar" controls whether or not Privoxy will appear as a button on the Task bar when minimized:

```
show-on-task-bar 0
```

If "close-button-minimizes" is set to 1, the Windows close button will minimize Privoxy instead of closing the program (close with the exit option on the File menu).

```
close-button-minimizes 1
```

The "hide-console" option is specific to the MS-Win console version of Privoxy. If this option is used, Privoxy will disconnect from and hide the command console.

```
#hide-console
```

---

8. Actions Files

The actions files are used to define what *actions* Privoxy takes for which URLs, and thus determines how ad images, cookies and various other aspects of HTTP content and transactions are handled, and on which sites (or even parts thereof). There are a number of such actions, with a wide range of functionality. Each action does something a little different. These actions give us a veritable arsenal of tools with which to exert our control, preferences and independence. Actions can be combined so that their effects are aggregated when applied against a given set of URLs.

There are three action files included with Privoxy with differing purposes:

- `default.action` - is the primary action file that sets the initial values for all actions. It is intended to provide a base level of functionality for Privoxy's array of features. So it is a set of broad rules that should work reasonably well as-is for most users. This is the file that the developers are keeping updated, and making available to users. The user's preferences as set in `standard.action`, e.g. either `Cautious` (the default), `Medium`, or `Advanced` (see below).
- `user.action` - is intended to be for local site preferences and exceptions. As an example, if your ISP or your bank has specific requirements, and need special handling, this kind of thing should go here. This file will not be upgraded.
- `standard.action` - is used only by the web based editor at http://config.privoxy.org/edit-actions-list?f=default, to set various pre-defined sets of rules for the default actions section in `default.action`.

  Edit Set to Cautious Set to Medium Set to Advanced

  These have increasing levels of aggressiveness *and have no influence on your browsing unless you select them explicitly in the editor*. A default installation should be pre-set to `Cautious` (versions prior to 3.0.5 were set to `Medium`). New users should try this for a while before adjusting the settings to more aggressive levels. The more aggressive the settings, then the more likelihood there is of problems such as sites not working as they should.

  The Edit button allows you to turn each action on/off individually for fine-tuning. The Cautious button changes the actions list to low/safe settings which will activate ad blocking and a minimal set of Privoxy's features, and subsequently there will be less of a chance for accidental problems. The Medium button sets the list to a medium level of other features and a low level set of privacy features. The Advanced button sets the list to a high level of ad blocking and medium level of privacy. See the chart below. The latter three buttons over-ride any changes via with the Edit button. More fine-tuning can be done in the lower sections of this internal page.

  It is not recommend to edit the `standard.action` file itself.

  The default profiles, and their associated actions, as pre-defined in `standard.action` are:

Table 1. Default Configurations

Feature	Cautious	Medium	Advanced
Ad-blocking Aggressiveness	medium	high	high
Ad-filtering by size	no	yes	yes
Ad-filtering by link	no	no	yes
Pop-up killing	blocks only	blocks only	blocks only
Privacy Features	low	medium	medium/high
Cookie handling	none	session-only	kill
Referer forging	no	yes	yes
GIF de-animation	no	yes	yes
Fast redirects	no	no	yes
HTML taming	no	no	yes
JavaScript taming	no	no	yes
Web-bug killing	no	yes	yes
Image tag reordering	no	no	yes

The list of actions files to be used are defined in the main configuration file, and are processed in the order they are defined (e.g. `default.action` is typically processed before `user.action`). The content of these can all be viewed and edited from http://config.privoxy.org/show-status. The over-riding principle when applying actions, is that the last action that matches a given URL wins. The broadest, most general rules go first (defined in `default.action`), followed by any exceptions (typically also in `default.action`), which are then followed lastly by any local preferences (typically in *user*`.action`). Generally, `user.action` has the last word.

An actions file typically has multiple sections. If you want to use "aliases" in an actions file, you have to place the (optional) alias section at the top of that file. Then comes the default set of rules which will apply universally to all sites and pages (be *very careful* with using

such a universal set in `user.action` or any other actions file after `default.action`, because it will override the result from consulting any previous file). And then below that, exceptions to the defined universal policies. You can regard `user.action` as an appendix to `default.action`, with the advantage that it is a separate file, which makes preserving your personal settings across Privoxy upgrades easier.

Actions can be used to block anything you want, including ads, banners, or just some obnoxious URL whose content you would rather not see. Cookies can be accepted or rejected, or accepted only during the current browser session (i.e. not written to disk), content can be modified, some JavaScripts tamed, user-tracking fooled, and much more. See below for a complete list of actions.

8.1. Finding the Right Mix

Note that some actions, like cookie suppression or script disabling, may render some sites unusable that rely on these techniques to work properly. Finding the right mix of actions is not always easy and certainly a matter of personal taste. And, things can always change, requiring refinements in the configuration. In general, it can be said that the more "aggressive" your default settings (in the top section of the actions file) are, the more exceptions for "trusted" sites you will have to make later. If, for example, you want to crunch all cookies per default, you'll have to make exceptions from that rule for sites that you regularly use and that require cookies for actually useful purposes, like maybe your bank, favorite shop, or newspaper.

We have tried to provide you with reasonable rules to start from in the distribution actions files. But there is no general rule of thumb on these things. There just are too many variables, and sites are constantly changing. Sooner or later you will want to change the rules (and read this chapter again :).

8.2. How to Edit

The easiest way to edit the actions files is with a browser by using our browser-based editor, which can be reached from http://config.privoxy.org/show-status. Note: the config file option enable-edit-actions must be enabled for this to work. The editor allows both fine-grained control over every single feature on a per-URL basis, and easy choosing from wholesale sets of defaults like "Cautious", "Medium" or "Advanced". Warning: the "Advanced" setting is more aggressive, and will be more likely to cause problems for some sites. Experienced users only!

If you prefer plain text editing to GUIs, you can of course also directly edit the the actions files with your favorite text editor. Look at `default.action` which is richly commented with many good examples.

8.3. How Actions are Applied to Requests

Actions files are divided into sections. There are special sections, like the "alias" sections which will be discussed later. For now let's concentrate on regular sections: They have a heading line (often split up to multiple lines for readability) which consist of a list of actions, separated by whitespace and enclosed in curly braces. Below that, there is a list of URL and tag patterns, each on a separate line.

To determine which actions apply to a request, the URL of the request is compared to all URL patterns in each "action file". Every time it matches, the list of applicable actions for the request is incrementally updated, using the heading of the section in which the pattern is located. The same is done again for tags and tag patterns later on.

If multiple applying sections set the same action differently, the last match wins. If not, the effects are aggregated. E.g. a URL might match a regular section with a heading line of `{ +handle-as-image }`, then later another one with just `{ +block }`, resulting in *both* actions to apply. And there may well be cases where you will want to combine actions together. Such a section then might look like:

```
{ +handle-as-image  +block }
# Block these as if they were images. Send no block page.
 banners.example.com
 media.example.com/.*banners
 .example.com/images/ads/
```

You can trace this process for URL patterns and any given URL by visiting http://config.privoxy.org/show-url-info.

Examples and more detail on this is provided in the Appendix, Troubleshooting: Anatomy of an Action section.

8.4. Patterns

As mentioned, Privoxy uses "patterns" to determine what *actions* might apply to which sites and pages your browser attempts to access. These "patterns" use wild card type *pattern* matching to achieve a high degree of flexibility. This allows one expression to be expanded and potentially match against many similar patterns.

Generally, an URL pattern has the form `<domain>/<path>`, where both the `<domain>` and `<path>` are optional. (This is why the special `/` pattern matches all URLs). Note that the protocol portion of the URL pattern (e.g. `http://`) should *not* be included in the pattern. This is assumed already!

The pattern matching syntax is different for the domain and path parts of the URL. The domain part uses a simple globbing type matching technique, while the path part uses a more flexible "Regular Expressions (PCRE)" based syntax.

`www.example.com/`
: is a domain-only pattern and will match any request to `www.example.com`, regardless of which document on that server is requested. So ALL pages in this domain would be covered by the scope of this action. Note that a simple `example.com` is different and would NOT match.

`www.example.com`
: means exactly the same. For domain-only patterns, the trailing `/` may be omitted.

`www.example.com/index.html$`
: matches all the documents on `www.example.com` whose name starts with `/index.html`.

`www.example.com/index.html$`
: matches only the single document `/index.html` on `www.example.com`.

`/index.html$`
: matches the document `/index.html`, regardless of the domain, i.e. on *any* web server anywhere.

`index.html`
: matches nothing, since it would be interpreted as a domain name and there is no top-level domain called `.html`. So its a mistake.

8.4.1. The Domain Pattern

The matching of the domain part offers some flexible options: if the domain starts or ends with a dot, it becomes unanchored at that end. For example:

`.example.com`
: matches any domain with first-level domain `com` and second-level domain `example`. For example `www.example.com`, `example.com` and `foo.bar.baz.example.com`. Note that it wouldn't match if the second-level domain was `another-example`.

`www.`
: matches any domain that *STARTS* with `www.` (It also matches the domain `www` but most of the time that doesn't matter.)

`.example.`
: matches any domain that *CONTAINS* `.example.`. And, by the way, also included would be any files or documents that exist within that domain since no path limitations are specified. (Correctly speaking: It matches any FQDN that contains `example` as a domain.) This might be `www.example.com`, `news.example.de`, or `www.example.net/cgi/testing.pl` for instance. All these cases are matched.

Additionally, there are wild-cards that you can use in the domain names themselves. These work similarly to shell globbing type wild-cards: "*" represents zero or more arbitrary characters (this is equivalent to the "Regular Expression" based syntax of ".*"), "?" represents any single character (this is equivalent to the regular expression syntax of a simple "."), and you can define "character classes" in square brackets which is similar to the same regular expression technique. All of this can be freely mixed:

`ad*.example.com`
: matches "adserver.example.com", "ads.example.com", etc but not "sfads.example.com"

`*ad*.example.com`
: matches all of the above, and then some.

`.?pix.com`
: matches `www.ipix.com`, `pictures.epix.com`, `a.b.c.d.e.upix.com` etc.

`www[1-9a-ez].example.c*`
: matches `www1.example.com`, `www4.example.cc`, `wwwd.example.cy`, `wwwz.example.com` etc., but *not* `wwww.example.com`.

While flexible, this is not the sophistication of full regular expression based syntax.

8.4.2. The Path Pattern

Privoxy uses Perl compatible (PCRE) "Regular Expression" based syntax (through the PCRE library) for matching the path portion (after the slash), and is thus more flexible.

There is an Appendix with a brief quick-start into regular expressions, and full (very technical) documentation on PCRE regex syntax is available on-line at http://www.pcre.org/man.txt. You might also find the Perl man page on regular expressions (`man perlre`) useful, which is available on-line at http://perldoc.perl.org/perlre.html.

Note that the path pattern is automatically left-anchored at the "/", i.e. it matches as if it would start with a "^" (regular expression speak for the beginning of a line).

Please also note that matching in the path is *CASE INSENSITIVE* by default, but you can switch to case sensitive at any point in the pattern by using the "(?-i)" switch: `www.example.com/(?-i)PaTtErN.*` will match only documents whose path starts with `PaTtErN` in *exactly* this capitalization.

`.example.com/.*`
: Is equivalent to just ".example.com", since any documents within that domain are matched with or without the ".*" regular expression. This is redundant

`.example.com/.*/index.html$`
: Will match any page in the domain of "example.com" that is named "index.html", and that is part of some path. For example, it matches "www.example.com/testing/index.html" but NOT "www.example.com/index.html" because the regular expression called for at least two "/'s", thus the path requirement. It also would match "www.example.com/testing/index_html", because of the special meta-character ".".

`.example.com/(.*/)?index\.html$`
: This regular expression is conditional so it will match any page named "index.html" regardless of path which in this case can have one or more "/'s". And this one must contain exactly ".html" (but does not have to end with that!).

`.example.com/(.*/)(ads|banners?|junk)`
: This regular expression will match any path of "example.com" that contains any of the words "ads", "banner", "banners" (because of the "?") or "junk". The path does not have to end in these words, just contain them.

`.example.com/(.*/)(ads|banners?|junk)/.*\.(jpe?g|gif|png)$`
: This is very much the same as above, except now it must end in either ".jpg", ".jpeg", ".gif" or ".png". So this one is limited to common image formats.

There are many, many good examples to be found in `default.action`, and more tutorials below in Appendix on regular expressions.

8.4.3. The Tag Pattern

Tag patterns are used to change the applying actions based on the request's tags. Tags can be created with either the client-header-tagger or the server-header-tagger action.

Tag patterns have to start with "TAG:", so Privoxy can tell them apart from URL patterns. Everything after the colon including white space, is interpreted as a regular expression with path pattern syntax, except that tag patterns aren't left-anchored automatically (Privoxy doesn't silently add a "^", you have to do it yourself if you need it).

To match all requests that are tagged with "foo" your pattern line should be "TAG:^foo$", "TAG:foo" would work as well, but it would also match requests whose tags contain "foo" somewhere. "TAG: foo" wouldn't work as it requires white space.

Sections can contain URL and tag patterns at the same time, but tag patterns are checked after the URL patterns and thus always overrule them, even if they are located before the URL patterns.

Once a new tag is added, Privoxy checks right away if it's matched by one of the tag patterns and updates the action settings accordingly. As a result tags can be used to activate other tagger actions, as long as these other taggers look for headers that haven't already be parsed.

For example you could tag client requests which use the `POST` method, then use this tag to activate another tagger that adds a tag if cookies are sent, and then use a block action based on the cookie tag. This allows the outcome of one action, to be input into a subsequent action. However if you'd reverse the position of the described taggers, and activated the method tagger based on the cookie tagger, no method tags would be created. The method tagger would look for the request line, but at the time the cookie tag is created, the request line has already been parsed.

While this is a limitation you should be aware of, this kind of indirection is seldom needed anyway and even the example doesn't make too much sense.

8.5. Actions

All actions are disabled by default, until they are explicitly enabled somewhere in an actions file. Actions are turned on if preceded with a "+", and turned off if preceded with a "-". So a `+action` means "do that action", e.g. `+block` means "please block URLs that match the following patterns", and `-block` means "don't block URLs that match the following patterns, even if `+block` previously applied."

Again, actions are invoked by placing them on a line, enclosed in curly braces and separated by whitespace, like in `{+some-action -some-other-action{some-parameter}}`, followed by a list of URL patterns, one per line, to which they apply. Together, the actions line and the following pattern lines make up a section of the actions file.

Actions fall into three categories:

- Boolean, i.e the action can only be "enabled" or "disabled". Syntax:

```
+name        # enable action name
-name        # disable action name
```

  Example: `+block`
- Parameterized, where some value is required in order to enable this type of action. Syntax:

```
+name{param}  # enable action and set parameter to param,
              # overwriting parameter from previous match if necessary
-name          # disable action. The parameter can be omitted
```

  Note that if the URL matches multiple positive forms of a parameterized action, the last match wins, i.e. the params from earlier matches are simply ignored.

Example: `+hide-user-agent{Mozilla/5.0 (X11; U; FreeBSD i386; en-US; rv:1.8.1.4) Gecko/20070602 Firefox/2.0.0.4}`

- Multi-value. These look exactly like parameterized actions, but they behave differently: If the action applies multiple times to the same URL, but with different parameters, *all* the parameters from *all* matches are remembered. This is used for actions that can be executed for the same request repeatedly, like adding multiple headers, or filtering through multiple filters. Syntax:

```
+name{param}   # enable action and add param to the list of parameters
-name{param}   # remove the parameter param from the list of parameters
              # If it was the last one left, disable the action.
-name          # disable this action completely and remove all parameters from the list
```

Examples: `+add-header{X-Fun-Header: Some text}` and `+filter{html-annoyances}`

If nothing is specified in any actions file, no "actions" are taken. So in this case Privoxy would just be a normal, non-blocking, non-filtering proxy. You must specifically enable the privacy and blocking features you need (although the provided default actions files will give a good starting point).

Later defined action sections always over-ride earlier ones of the same type. So exceptions to any rules you make, should come in the latter part of the file (or in a file that is processed later when using multiple actions files such as `user.action`). For multi-valued actions, the actions are applied in the order they are specified. Actions files are processed in the order they are defined in `config` (the default installation has three actions files). It also quite possible for any given URL to match more than one "pattern" (because of wildcards and regular expressions), and thus to trigger more than one set of actions! Last match wins.

The list of valid Privoxy actions are:

8.5.1. add-header

Typical use:
: Confuse log analysis, custom applications

Effect:
: Sends a user defined HTTP header to the web server.

Type:
: Multi-value.

Parameter:
: Any string value is possible. Validity of the defined HTTP headers is not checked. It is recommended that you use the "`X-`" prefix for custom headers.

Notes:
: This action may be specified multiple times, in order to define multiple headers. This is rarely needed for the typical user. If you don't know what "HTTP headers" are, you definitely don't need to worry about this one.

Example usage:

```
+add-header{X-User-Tracking: sucks}
```

8.5.2. block

Typical use:
: Block ads or other unwanted content

Effect:
: Requests for URLs to which this action applies are blocked, i.e. the requests are trapped by Privoxy and the requested URL is never retrieved, but is answered locally with a substitute page or image, as determined by the `handle-as-image`, `set-image-blocker`, and `handle-as-empty-document` actions.

Type:
: Boolean.

Parameter:
: N/A

Notes:
: Privoxy sends a special "BLOCKED" page for requests to blocked pages. This page contains links to find out why the request was blocked, and a click-through to the blocked content (the latter only if compiled with the force feature enabled). The "BLOCKED" page adapts to the available screen space -- it displays full-blown if space allows, or miniaturized and text-only if loaded into a small frame or window. If you are using Privoxy right now, you can take a look at the "BLOCKED" page.

  A very important exception occurs if *both* `block` and `handle-as-image`, apply to the same request: it will then be replaced by an image. If `set-image-blocker` (see below) also applies, the type of image will be determined by its parameter, if not, the standard checkerboard pattern is sent.

  It is important to understand this process, in order to understand how Privoxy deals with ads and other unwanted content. Blocking is a core feature, and one upon which various other features depend.

  The `filter` action can perform a very similar task, by "blocking" banner images and other content through rewriting the relevant URLs in the document's HTML source, so they don't get requested in the first place. Note that this is a totally different technique, and it's easy to confuse the two.

Example usage (section):

```
{+block}
# Block and replace with "blocked" page
 .nasty-stuff.example.com

{+block +handle-as-image}
# Block and replace with image
 .ad.doubleclick.net
 .ads.r.us/banners/

{+block +handle-as-empty-document}
# Block and then ignore
 adserver.exampleclick.net/.*\.js$
```

8.5.3. client-header-filter

Typical use:
: Rewrite or remove single client headers.

Effect:
: All client headers to which this action applies are filtered on-the-fly through the specified regular expression based substitutions.

Type:
: Parameterized.

Parameter:
: The name of a client-header filter, as defined in one of the filter files.

Notes:
: Client-header filters are applied to each header on its own, not to all at once. This makes it easier to diagnose problems, but on the downside you can't write filters that only change header x if header y's value is z. You can do that by using tags though.

  Client-header filters are executed after the other header actions have finished and use their output as input.

  If the request URL gets changed, Privoxy will detect that and use the new one. This can be used to rewrite the request destination behind the client's back, for example to specify a Tor exit relay for certain requests.

  Please refer to the filter file chapter to learn which client-header filters are available by default, and how to create your own.

Example usage (section):

```
{+client-header-filter{hide-tor-exit-notation}}
.exit/
```

8.5.4. client-header-tagger

Typical use:
: Block requests based on their headers.

Effect:
: Client headers to which this action applies are filtered on-the-fly through the specified regular expression based substitutions, the result is used as tag.

Type:
: Parameterized.

Parameter:
: The name of a client-header tagger, as defined in one of the filter files.

Notes:
: Client-header taggers are applied to each header on its own, and as the header isn't modified, each tagger "sees" the original.

  Client-header taggers are the first actions that are executed and their tags can be used to control every other action.

Example usage (section):

```
# Tag every request with the User-Agent header
{+client-header-tagger{user-agent}}
/
```

8.5.5. content-type-overwrite

Typical use:
: Stop useless download menus from popping up, or change the browser's rendering mode

Effect:
: Replaces the "Content-Type:" HTTP server header.

Type:
: Parameterized.

Parameter:

Any string.

Notes:

The "Content-Type:" HTTP server header is used by the browser to decide what to do with the document. The value of this header can cause the browser to open a download menu instead of displaying the document by itself, even if the document's format is supported by the browser.

The declared content type can also affect which rendering mode the browser chooses. If XHTML is delivered as "text/html", many browsers treat it as yet another broken HTML document. If it is send as "application/xml", browsers with XHTML support will only display it, if the syntax is correct.

If you see a web site that proudly uses XHTML buttons, but sets "Content-Type: text/html", you can use Privoxy to overwrite it with "application/xml" and validate the web master's claim inside your XHTML-supporting browser. If the syntax is incorrect, the browser will complain loudly.

You can also go the opposite direction: if your browser prints error messages instead of rendering a document falsely declared as XHTML, you can overwrite the content type with "text/html" and have it rendered as broken HTML document.

By default `content-type-overwrite` only replaces "Content-Type:" headers that look like some kind of text. If you want to overwrite it unconditionally, you have to combine it with `force-text-mode`. This limitation exists for a reason, think twice before circumventing it.

Most of the time it's easier to replace this action with a custom `server-header filter`. It allows you to activate it for every document of a certain site and it will still only replace the content types you aimed at.

Of course you can apply `content-type-overwrite` to a whole site and then make URL based exceptions, but it's a lot more work to get the same precision.

Example usage (sections):

```
# Check if www.example.net/ really uses valid XHTML
{ +content-type-overwrite{application/xml} }
www.example.net/

# but leave the content type unmodified if the URL looks like a style sheet
{-content-type-overwrite}
www.example.net/.*\.css$
www.example.net/.*style
```

8.5.6. crunch-client-header

Typical use:

Remove a client header Privoxy has no dedicated action for.

Effect:

Deletes every header sent by the client that contains the string the user supplied as parameter.

Type:

Parameterized.

Parameter:

Any string.

Notes:

This action allows you to block client headers for which no dedicated Privoxy action exists. Privoxy will remove every client header that contains the string you supplied as parameter.

Regular expressions are *not supported* and you can't use this action to block different headers in the same request, unless they contain the same string.

`crunch-client-header` is only meant for quick tests. If you have to block several different headers, or only want to modify parts of them, you should use a `client-header filter`.

Warning
Don't block any header without understanding the consequences.

Example usage (section):

```
# Block the non-existent "Privacy-Violation:" client header
{ +crunch-client-header{Privacy-Violation:} }
/
```

8.5.7. crunch-if-none-match

Typical use:
: Prevent yet another way to track the user's steps between sessions.

Effect:
: Deletes the "If-None-Match:" HTTP client header.

Type:
: Boolean.

Parameter:
: N/A

Notes:
: Removing the "If-None-Match:" HTTP client header is useful for filter testing, where you want to force a real reload instead of getting status code "304" which would cause the browser to use a cached copy of the page.

  It is also useful to make sure the header isn't used as a cookie replacement (unlikely but possible).

  Blocking the "If-None-Match:" header shouldn't cause any caching problems, as long as the "If-Modified-Since:" header isn't blocked or missing as well.

  It is recommended to use this action together with `hide-if-modified-since` and `overwrite-last-modified`.

Example usage (section):

```
# Let the browser revalidate cached documents but don't
# allow the server to use the revalidation headers for user tracking.
{+hide-if-modified-since{-60} \
 +overwrite-last-modified{randomize} \
 +crunch-if-none-match}
/
```

8.5.8. crunch-incoming-cookies

Typical use:
: Prevent the web server from setting HTTP cookies on your system

Effect:
: Deletes any "Set-Cookie:" HTTP headers from server replies.

Type:
: Boolean.

Parameter:
: N/A

Notes:
: This action is only concerned with *incoming* HTTP cookies. For *outgoing* HTTP cookies, use `crunch-outgoing-cookies`. Use *both* to disable HTTP cookies completely.

  It makes *no sense at all* to use this action in conjunction with the `session-cookies-only` action, since it would prevent the session cookies from being set. See also `filter-content-cookies`.

Example usage:

```
+crunch-incoming-cookies
```

8.5.9. crunch-server-header

Typical use:
: Remove a server header Privoxy has no dedicated action for.

Effect:
: Deletes every header sent by the server that contains the string the user supplied as parameter.

Type:
: Parameterized.

Parameter:
: Any string.

Notes:
: This action allows you to block server headers for which no dedicated Privoxy action exists. Privoxy will remove every server header that contains the string you supplied as parameter.

  Regular expressions are *not supported* and you can't use this action to block different headers in the same request, unless they contain the same string.

  `crunch-server-header` is only meant for quick tests. If you have to block several different headers, or only want to modify parts of them, you should use a custom `server-header filter`.

Warning

Don't block any header without understanding the consequences.

Example usage (section):

```
# Crunch server headers that try to prevent caching
{ +crunch-server-header{no-cache} }
/
```

8.5.10. crunch-outgoing-cookies

Typical use:
: Prevent the web server from reading any HTTP cookies from your system

Effect:
: Deletes any "Cookie:" HTTP headers from client requests.

Type:
: Boolean.

Parameter:
: N/A

Notes:
: This action is only concerned with *outgoing* HTTP cookies. For *incoming* HTTP cookies, use `crunch-incoming-cookies`. Use *both* to disable HTTP cookies completely.

  It makes *no sense at all* to use this action in conjunction with the `session-cookies-only` action, since it would prevent the session cookies from being read.

Example usage:

```
+crunch-outgoing-cookies
```

8.5.11. deanimate-gifs

Typical use:
: Stop those annoying, distracting animated GIF images.

Effect:
: De-animate GIF animations, i.e. reduce them to their first or last image.

Type:
: Parameterized.

Parameter:
: "last" or "first"

Notes:
: This will also shrink the images considerably (in bytes, not pixels!). If the option "first" is given, the first frame of the animation is used as the replacement. If "last" is given, the last frame of the animation is used instead, which probably makes more sense for most banner animations, but also has the risk of not showing the entire last frame (if it is only a delta to an earlier frame).

  You can safely use this action with patterns that will also match non-GIF objects, because no attempt will be made at anything that doesn't look like a GIF.

Example usage:

```
+deanimate-gifs{last}
```

8.5.12. downgrade-http-version

Typical use:
: Work around (very rare) problems with HTTP/1.1

Effect:
: Downgrades HTTP/1.1 client requests and server replies to HTTP/1.0.

Type:
: Boolean.

Parameter:
: N/A

Notes:
: This is a left-over from the time when Privoxy didn't support important HTTP/1.1 features well. It is left here for the unlikely case that you experience HTTP/1.1 related problems with some server out there. Not all HTTP/1.1 features and requirements are supported yet, so there is a chance you might need this action.

Example usage (section):

```
{+downgrade-http-version}
problem-host.example.com
```

8.5.13. fast-redirects

Typical use:

Fool some click-tracking scripts and speed up indirect links.

Effect:

Detects redirection URLs and redirects the browser without contacting the redirection server first.

Type:

Parameterized.

Parameter:

◊ "simple-check" to just search for the string "http://" to detect redirection URLs.
◊ "check-decoded-url" to decode URLs (if necessary) before searching for redirection URLs.

Notes:

Many sites, like yahoo.com, don't just link to other sites. Instead, they will link to some script on their own servers, giving the destination as a parameter, which will then redirect you to the final target. URLs resulting from this scheme typically look like: "http://www.example.org/click-tracker.cgi?target=http%3a//www.example.net/".

Sometimes, there are even multiple consecutive redirects encoded in the URL. These redirections via scripts make your web browsing more traceable, since the server from which you follow such a link can see where you go to. Apart from that, valuable bandwidth and time is wasted, while your browser asks the server for one redirect after the other. Plus, it feeds the advertisers.

This feature is currently not very smart and is scheduled for improvement. If it is enabled by default, you will have to create some exceptions to this action. It can lead to failures in several ways:

Not every URLs with other URLs as parameters is evil. Some sites offer a real service that requires this information to work. For example a validation service needs to know, which document to validate. `fast-redirects` assumes that every URL parameter that looks like another URL is a redirection target, and will always redirect to the last one. Most of the time the assumption is correct, but if it isn't, the user gets redirected anyway.

Another failure occurs if the URL contains other parameters after the URL parameter. The URL: "http://www.example.org/?redirect=http%3a//www.example.net/&foo=bar". contains the redirection URL "http://www.example.net/", followed by another parameter. `fast-redirects` doesn't know that and will cause a redirect to "http://www.example.net/&foo=bar". Depending on the target server configuration, the parameter will be silently ignored or lead to a "page not found" error. You can prevent this problem by first using the `redirect` action to remove the last part of the URL, but it requires a little effort.

To detect a redirection URL, `fast-redirects` only looks for the string "http://", either in plain text (invalid but often used) or encoded as "http%3a//". Some sites use their own URL encoding scheme, encrypt the address of the target server or replace it with a database id. In theses cases `fast-redirects` is fooled and the request reaches the redirection server where it probably gets logged.

Example usage:

```
{ +fast-redirects{simple-check} }
  one.example.com

{ +fast-redirects{check-decoded-url} }
  another.example.com/testing
```

8.5.14. filter

Typical use:

Get rid of HTML and JavaScript annoyances, banner advertisements (by size), do fun text replacements, add personalized effects, etc.

Effect:

All instances of text-based type, most notably HTML and JavaScript, to which this action applies, can be filtered on-the-fly through the specified regular expression based substitutions. (Note: as of version 3.0.3 plain text documents are exempted from filtering, because web servers often use the `text/plain` MIME type for all files whose type they don't know.)

Type:

Parameterized.

Parameter:

The name of a content filter, as defined in the filter file. Filters can be defined in one or more files as defined by the `filterfile` option in the config file. `default.filter` is the collection of filters supplied by the developers. Locally defined filters should go in their own file, such as `user.filter`.

When used in its negative form, and without parameters, *all* filtering is completely disabled.

Notes:

For your convenience, there are a number of pre-defined filters available in the distribution filter file that you can use. See the examples below for a list.

Filtering requires buffering the page content, which may appear to slow down page rendering since nothing is displayed until all content has passed the filters. (It does not really take longer, but seems that way since the page is not incrementally displayed.) This effect will be more noticeable on slower connections.

"Rolling your own" filters requires a knowledge of "Regular Expressions" and "HTML". This is very powerful feature, and potentially very intrusive. Filters should be used with caution, and where an equivalent "action" is not available.

The amount of data that can be filtered is limited to the `buffer-limit` option in the main config file. The default is 4096 KB (4 Megs). Once this limit is exceeded, the buffered data, and all pending data, is passed through unfiltered.

Inappropriate MIME types, such as zipped files, are not filtered at all. (Again, only text-based types except plain text). Encrypted SSL data (from HTTPS servers) cannot be filtered either, since this would violate the integrity of the secure transaction. In some situations it might be necessary to protect certain text, like source code, from filtering by defining appropriate `-filter` exceptions.

Compressed content can't be filtered either, unless Privoxy is compiled with zlib support (requires at least Privoxy 3.0.7), in which case Privoxy will decompress the content before filtering it.

If you use a Privoxy version without zlib support, but want filtering to work on as much documents as possible, even those that would normally be sent compressed, you must use the `prevent-compression` action in conjunction with `filter`.

Content filtering can achieve some of the same effects as the `block` action, i.e. it can be used to block ads and banners. But the mechanism works quite differently. One effective use, is to block ad banners based on their size (see below), since many of these seem to be somewhat standardized.

Feedback with suggestions for new or improved filters is particularly welcome!

The below list has only the names and a one-line description of each predefined filter. There are more verbose explanations of what these filters do in the filter file chapter.

Example usage (with filters from the distribution `default.filter` file). See the Predefined Filters section for more explanation on each:

```
+filter{js-annoyances}       # Get rid of particularly annoying JavaScript abuse

+filter{js-events}           # Kill all JS event bindings (Radically destructive! Only for extra nasty sites)

+filter{html-annoyances}     # Get rid of particularly annoying HTML abuse

+filter{content-cookies}     # Kill cookies that come in the HTML or JS content

+filter{refresh-tags}        # Kill automatic refresh tags (for dial-on-demand setups)

+filter{unsolicited-popups}  # Disable only unsolicited pop-up windows. Useful if your browser lacks this ability.

+filter{all-popups}          # Kill all popups in JavaScript and HTML. Useful if your browser lacks this ability.

+filter{img-reorder}         # Reorder attributes in <img> tags to make the banners-by-* filters more effective

+filter{banners-by-size}     # Kill banners by size

+filter{banners-by-link}     # Kill banners by their links to known clicktrackers

+filter{webbugs}             # Squish WebBugs (1x1 invisible GIFs used for user tracking)

+filter{tiny-textforms}      # Extend those tiny textareas up to 40x80 and kill the hard wrap

+filter{jumping-windows}     # Prevent windows from resizing and moving themselves

+filter{frameset-borders}    # Give frames a border and make them resizeable

+filter{demoronizer}         # Fix MS's non-standard use of standard charsets

+filter{shockwave-flash}     # Kill embedded Shockwave Flash objects

+filter{quicktime-kioskmode} # Make Quicktime movies savable

+filter{fun}                 # Text replacements for subversive browsing fun!

```

```
+filter{crude-parental}      # Crude parental filtering (demo only)

+filter{ie-exploits}         # Disable a known Internet Explorer bug exploits

+filter{site-specifics}      # Custom filters for specific site related problems

+filter{google}              # Removes text ads and other Google specific improvements

+filter{yahoo}               # Removes text ads and other Yahoo specific improvements

+filter{msn}                 # Removes text ads and other MSN specific improvements

+filter{blogspot}            # Cleans up Blogspot blogs

+filter{no-ping}             # Removes non-standard ping attributes from anchor and area tags
```

8.5.15. force-text-mode

Typical use:

Force Privoxy to treat a document as if it was in some kind of *text* format.

Effect:

Declares a document as text, even if the "Content-Type:" isn't detected as such.

Type:

Boolean.

Parameter:

N/A

Notes:

As explained above, Privoxy tries to only filter files that are in some kind of text format. The same restrictions apply to `content-type-overwrite`. `force-text-mode` declares a document as text, without looking at the "Content-Type:" first.

Warning
Think twice before activating this action. Filtering binary data with regular expressions can cause file damage.

Example usage:

```
+force-text-mode
```

8.5.16. forward-override

Typical use:

Change the forwarding settings based on User-Agent or request origin

Effect:

Overrules the forward directives in the configuration file.

Type:

Multi-value.

Parameter:

- "forward ." to use a direct connection without any additional proxies.
- "forward 127.0.0.1:8123" to use the HTTP proxy listening at 127.0.0.1 port 8123.
- "forward-socks4a 127.0.0.1:9050 ." to use the socks4a proxy listening at 127.0.0.1 port 9050. Replace "forward-socks4a" with "forward-socks4" to use a socks4 connection (with local DNS resolution) instead.
- "forward-socks4a 127.0.0.1:9050 proxy.example.org:8000" to use the socks4a proxy listening at 127.0.0.1 port 9050 to reach the HTTP proxy listening at proxy.example.org port 8000. Replace "forward-socks4a" with "forward-socks4" to use a socks4 connection (with local DNS resolution) instead.

Notes:

This action takes parameters similar to the forward directives in the configuration file, but without the URL pattern. It can be used as replacement, but normally it's only used in cases where matching based on the request URL isn't sufficient.

Warning
Please read the description for the forward directives before using this action. Forwarding to the wrong people will reduce your privacy and increase the chances of man-in-the-middle attacks. If the ports are missing or invalid, default values will be used. This might change in the future and you shouldn't rely on it. Otherwise incorrect syntax causes Privoxy to exit. Use the show-url-info CGI page to verify that your forward settings do what you thought the do.

Example usage:

```
# Always use direct connections for requests previously tagged as
# "User-Agent: fetch libfetch/2.0" and make sure
# resuming downloads continues to work.
# This way you can continue to use Tor for your normal browsing,
# without overloading the Tor network with your FreeBSD ports updates
# or downloads of bigger files like ISOs.
# Note that HTTP headers are easy to fake and therefore their
# values are as (un)trustworthy as your clients and users.
{+forward-override{forward .} \
 -hide-if-modified-since      \
 -overwrite-last-modified     \
}
TAG:^User-Agent: fetch libfetch/2\.0$
```

8.5.17. handle-as-empty-document

Typical use:

Mark URLs that should be replaced by empty documents *if they get blocked*

Effect:

This action alone doesn't do anything noticeable. It just marks URLs. If the `block` action *also applies*, the presence or absence of this mark decides whether an HTML "BLOCKED" page, or an empty document will be sent to the client as a substitute for the blocked content. The *empty* document isn't literally empty, but actually contains a single space.

Type:

Boolean.

Parameter:

N/A

Notes:

Some browsers complain about syntax errors if JavaScript documents are blocked with Privoxy's default HTML page; this option can be used to silence them. And of course this action can also be used to eliminate the Privoxy BLOCKED message in frames.

The content type for the empty document can be specified with `content-type-overwrite{}`, but usually this isn't necessary.

Example usage:

```
# Block all documents on example.org that end with ".js",
# but send an empty document instead of the usual HTML message.
{+block +handle-as-empty-document}
example.org/.*\.js$
```

8.5.18. handle-as-image

Typical use:

Mark URLs as belonging to images (so they'll be replaced by images *if they do get blocked*, rather than HTML pages)

Effect:

This action alone doesn't do anything noticeable. It just marks URLs as images. If the `block` action *also applies*, the presence or absence of this mark decides whether an HTML "blocked" page, or a replacement image (as determined by the `set-image-blocker` action) will be sent to the client as a substitute for the blocked content.

Type:

Boolean.

Parameter:

N/A

Notes:

The below generic example section is actually part of `default.action`. It marks all URLs with well-known image file name extensions as images and should be left intact.

Users will probably only want to use the handle-as-image action in conjunction with `block`, to block sources of banners, whose URLs don't reflect the file type, like in the second example section.

Note that you cannot treat HTML pages as images in most cases. For instance, (in-line) ad frames require an HTML page to be sent, or they won't display properly. Forcing `handle-as-image` in this situation will not replace the ad frame with an image, but lead to error messages.

Example usage (sections):

```
# Generic image extensions:
#
{+handle-as-image}
/.*\.(gif|jpg|jpeg|png|bmp|ico)$

# These don't look like images, but they're banners and should be
# blocked as images:
```

```
#
{+block +handle-as-image}
some.nasty-banner-server.com/junk.cgi\?output=trash

# Banner source! Who cares if they also have non-image content?
ad.doubleclick.net
```

8.5.19. hide-accept-language

Typical use:
: Pretend to use different language settings.

Effect:
: Deletes or replaces the "Accept-Language:" HTTP header in client requests.

Type:
: Parameterized.

Parameter:
: Keyword: "block", or any user defined value.

Notes:
: Faking the browser's language settings can be useful to make a foreign User-Agent set with `hide-user-agent` more believable.

  However some sites with content in different languages check the "Accept-Language:" to decide which one to take by default. Sometimes it isn't possible to later switch to another language without changing the "Accept-Language:" header first.

  Therefore it's a good idea to either only change the "Accept-Language:" header to languages you understand, or to languages that aren't wide spread.

  Before setting the "Accept-Language:" header to a rare language, you should consider that it helps to make your requests unique and thus easier to trace. If you don't plan to change this header frequently, you should stick to a common language.

Example usage (section):

```
# Pretend to use Canadian language settings.
{+hide-accept-language{en-ca} \
+hide-user-agent{Mozilla/5.0 (X11; U; OpenBSD i386; en-CA; rv:1.8.0.4) Gecko/20060628 Firefox/1.5.0.4} \
}
/
```

8.5.20. hide-content-disposition

Typical use:
: Prevent download menus for content you prefer to view inside the browser.

Effect:
: Deletes or replaces the "Content-Disposition:" HTTP header set by some servers.

Type:
: Parameterized.

Parameter:
: Keyword: "block", or any user defined value.

Notes:
: Some servers set the "Content-Disposition:" HTTP header for documents they assume you want to save locally before viewing them. The "Content-Disposition:" header contains the file name the browser is supposed to use by default.

  In most browsers that understand this header, it makes it impossible to *just view* the document, without downloading it first, even if it's just a simple text file or an image.

  Removing the "Content-Disposition:" header helps to prevent this annoyance, but some browsers additionally check the "Content-Type:" header, before they decide if they can display a document without saving it first. In these cases, you have to change this header as well, before the browser stops displaying download menus.

  It is also possible to change the server's file name suggestion to another one, but in most cases it isn't worth the time to set it up.

  This action will probably be removed in the future, use server-header filters instead.

Example usage:

```
# Disarm the download link in Sourceforge's patch tracker
{ -filter \
 +content-type-overwrite{text/plain}\
 +hide-content-disposition{block} }
 .sourceforge.net/tracker/download\.php
```

8.5.21. hide-if-modified-since

Typical use:
Prevent yet another way to track the user's steps between sessions.
Effect:
Deletes the "If-Modified-Since:" HTTP client header or modifies its value.
Type:
Parameterized.
Parameter:
Keyword: "block", or a user defined value that specifies a range of hours.
Notes:
Removing this header is useful for filter testing, where you want to force a real reload instead of getting status code "304", which would cause the browser to use a cached copy of the page.

Instead of removing the header, `hide-if-modified-since` can also add or subtract a random amount of time to/from the header's value. You specify a range of minutes where the random factor should be chosen from and Privoxy does the rest. A negative value means subtracting, a positive value adding.

Randomizing the value of the "If-Modified-Since:" makes it less likely that the server can use the time as a cookie replacement, but you will run into caching problems if the random range is too high.

It is a good idea to only use a small negative value and let `overwrite-last-modified` handle the greater changes.

It is also recommended to use this action together with `crunch-if-none-match`, otherwise it's more or less pointless.
Example usage (section):

```
# Let the browser revalidate but make tracking based on the time less likely.
{+hide-if-modified-since{-60} \
 +overwrite-last-modified{randomize} \
 +crunch-if-none-match}
/
```

8.5.22. hide-forwarded-for-headers

Typical use:
Improve privacy by not forwarding the source of the request in the HTTP headers.
Effect:
Deletes any existing "X-Forwarded-for:" HTTP header from client requests.
Type:
Boolean.
Parameter:
N/A
Notes:
It is safe and recommended to leave this on.
Example usage:

```
+hide-forwarded-for-headers
```

8.5.23. hide-from-header

Typical use:
Keep your (old and ill) browser from telling web servers your email address
Effect:
Deletes any existing "From:" HTTP header, or replaces it with the specified string.
Type:
Parameterized.
Parameter:
Keyword: "block", or any user defined value.
Notes:
The keyword "block" will completely remove the header (not to be confused with the `block` action).

Alternately, you can specify any value you prefer to be sent to the web server. If you do, it is a matter of fairness not to use any address that is actually used by a real person.

This action is rarely needed, as modern web browsers don't send "From:" headers anymore.
Example usage:

```
+hide-from-header{block}
```

or

```
```

```
+hide-from-header{spam-me-senseless@sittingduck.example.com}
```

8.5.24. hide-referrer

Typical use:

Conceal which link you followed to get to a particular site

Effect:

Deletes the "Referer:" (sic) HTTP header from the client request, or replaces it with a forged one.

Type:

Parameterized.

Parameter:

- ◊ "conditional-block" to delete the header completely if the host has changed.
- ◊ "conditional-forge" to forge the header if the host has changed.
- ◊ "block" to delete the header unconditionally.
- ◊ "forge" to pretend to be coming from the homepage of the server we are talking to.
- ◊ Any other string to set a user defined referrer.

Notes:

`conditional-block` is the only parameter, that isn't easily detected in the server's log file. If it blocks the referrer, the request will look like the visitor used a bookmark or typed in the address directly.

Leaving the referrer unmodified for requests on the same host allows the server owner to see the visitor's "click path", but in most cases she could also get that information by comparing other parts of the log file: for example the User-Agent if it isn't a very common one, or the user's IP address if it doesn't change between different requests.

Always blocking the referrer, or using a custom one, can lead to failures on servers that check the referrer before they answer any requests, in an attempt to prevent their content from being embedded or linked to elsewhere.

Both `conditional-block` and `forge` will work with referrer checks, as long as content and valid referring page are on the same host. Most of the time that's the case.

`hide-referer` is an alternate spelling of `hide-referrer` and the two can be can be freely substituted with each other. ("referrer" is the correct English spelling, however the HTTP specification has a bug - it requires it to be spelled as "referer".)

Example usage:

```
+hide-referrer{forge}
```

or

```
+hide-referrer{http://www.yahoo.com/}
```

8.5.25. hide-user-agent

Typical use:

Try to conceal your type of browser and client operating system

Effect:

Replaces the value of the "User-Agent:" HTTP header in client requests with the specified value.

Type:

Parameterized.

Parameter:

Any user-defined string.

Notes:

Warning
This can lead to problems on web sites that depend on looking at this header in order to customize their content for different browsers (which, by the way, is *NOT* the right thing to do: good web sites work browser-independently).

Using this action in multi-user setups or wherever different types of browsers will access the same Privoxy is *not recommended*. In single-user, single-browser setups, you might use it to delete your OS version information from the headers, because it is an invitation to exploit known bugs for your OS. It is also occasionally useful to forge this in order to access sites that won't let you in otherwise (though there may be a good reason in some cases). Example of this: some MSN sites will not let Mozilla enter, yet forging to a Netscape 6.1 user-agent works just fine. (Must be just a silly MS goof, I'm sure :-).

More information on known user-agent strings can be found at http://www.user-agents.org/ and http://en.wikipedia.org/wiki/User_agent.

Example usage:

```
+hide-user-agent{Netscape 6.1 (X11; I; Linux 2.4.18 i686)}
```

8.5.26. inspect-jpegs

Typical use:
: Try to protect against a MS buffer over-run in JPEG processing

Effect:
: Protect against a known exploit

Type:
: Boolean.

Parameter:
: N/A

Notes:
: See Microsoft Security Bulletin MS04-028. JPEG images are one of the most common image types found across the Internet. The exploit as described can allow execution of code on the target system, giving an attacker access to the system in question by merely planting an altered JPEG image, which would have no obvious indications of what lurks inside. This action tries to prevent this exploit if delivered through unencrypted HTTP.

  Note that the exploit mentioned is several years old and it's unlikely that your client is still vulnerable against it. This action may be removed in one of the next releases.

Example usage:

```
+inspect-jpegs
```

8.5.27. kill-popups

Typical use:
: Eliminate those annoying pop-up windows (deprecated)

Effect:
: While loading the document, replace JavaScript code that opens pop-up windows with (syntactically neutral) dummy code on the fly.

Type:
: Boolean.

Parameter:
: N/A

Notes:
: This action is basically a built-in, hardwired special-purpose filter action, but there are important differences: For `kill-popups`, the document need not be buffered, so it can be incrementally rendered while downloading. But `kill-popups` doesn't catch as many pop-ups as `filter{all-popups}` does and is not as smart as `filter{unsolicited-popups}` is.

  Think of it as a fast and efficient replacement for a filter that you can use if you don't want any filtering at all. Note that it doesn't make sense to combine it with any `filter` action, since as soon as one `filter` applies, the whole document needs to be buffered anyway, which destroys the advantage of the `kill-popups` action over its filter equivalent.

  Killing all pop-ups unconditionally is problematic. Many shops and banks rely on pop-ups to display forms, shopping carts etc, and the `filter{unsolicited-popups}` does a better job of catching only the unwanted ones.

  If the only kind of pop-ups that you want to kill are exit consoles (those *really nasty* windows that appear when you close an other one), you might want to use `filter{js-annoyances}` instead.

  This action is most appropriate for browsers that don't have any controls for unwanted pop-ups. Not recommended for general usage.

  This action doesn't work very reliable and may be removed in future releases.

Example usage:

```
+kill-popups
```

8.5.28. limit-connect

Typical use:
: Prevent abuse of Privoxy as a TCP proxy relay or disable SSL for untrusted sites

Effect:
: Specifies to which ports HTTP CONNECT requests are allowable.

Type:
: Parameterized.

Parameter:
: A comma-separated list of ports or port ranges (the latter using dashes, with the minimum defaulting to 0 and the maximum to 65K).

Notes:

By default, i.e. if no `limit-connect` action applies, Privoxy only allows HTTP CONNECT requests to port 443 (the standard, secure HTTPS port). Use `limit-connect` if more fine-grained control is desired for some or all destinations.

The CONNECT methods exists in HTTP to allow access to secure websites ("https://" URLs) through proxies. It works very simply: the proxy connects to the server on the specified port, and then short-circuits its connections to the client and to the remote server. This means CONNECT-enabled proxies can be used as TCP relays very easily.

Privoxy relays HTTPS traffic without seeing the decoded content. Websites can leverage this limitation to circumvent Privoxy's filters. By specifying an invalid port range you can disable HTTPS entirely. If you plan to disable SSL by default, consider enabling `treat-forbidden-connects-like-blocks` as well, to be able to quickly create exceptions.

Example usages:

```
+limit-connect{443}                  # This is the default and need not be specified.
+limit-connect{80,443}               # Ports 80 and 443 are OK.
+limit-connect{-3, 7, 20-100, 500-}  # Ports less than 3, 7, 20 to 100 and above 500 are OK.
+limit-connect{-}                    # All ports are OK
+limit-connect{,}                    # No HTTPS/SSL traffic is allowed
```

8.5.29. prevent-compression

Typical use:

Ensure that servers send the content uncompressed, so it can be passed through `filter`s.

Effect:

Removes the Accept-Encoding header which can be used to ask for compressed transfer.

Type:

Boolean.

Parameter:

N/A

Notes:

More and more websites send their content compressed by default, which is generally a good idea and saves bandwidth. But the `filter`, `deanimate-gifs` and `kill-popups` actions need access to the uncompressed data.

When compiled with zlib support (available since Privoxy 3.0.7), content that should be filtered is decompressed on-the-fly and you don't have to worry about this action. If you are using an older Privoxy version, or one that hasn't been compiled with zlib support, this action can be used to convince the server to send the content uncompressed.

Most text-based instances compress very well, the size is seldom decreased by less than 50%, for markup-heavy instances like news feeds saving more than 90% of the original size isn't unusual.

Not using compression will therefore slow down the transfer, and you should only enable this action if you really need it. As of Privoxy 3.0.7 it's disabled in all predefined action settings.

Note that some (rare) ill-configured sites don't handle requests for uncompressed documents correctly. Broken PHP applications tend to send an empty document body, some IIS versions only send the beginning of the content. If you enable `prevent-compression` per default, you might want to add exceptions for those sites. See the example for how to do that.

Example usage (sections):

```
# Selectively turn off compression, and enable a filter
#
{ +filter{tiny-textforms} +prevent-compression }
# Match only these sites
 .google.
 sourceforge.net
 sf.net

# Or instead, we could set a universal default:
#
{ +prevent-compression }
 / # Match all sites

# Then maybe make exceptions for broken sites:
#
{ -prevent-compression }
.compusa.com/
```

8.5.30. overwrite-last-modified

Typical use:

Prevent yet another way to track the user's steps between sessions.

Effect:

Deletes the "Last-Modified:" HTTP server header or modifies its value.

Type:

Parameterized.

Parameter:

One of the keywords: "block", "reset-to-request-time" and "randomize"

Notes:

Removing the "Last-Modified:" header is useful for filter testing, where you want to force a real reload instead of getting status code "304", which would cause the browser to reuse the old version of the page.

The "randomize" option overwrites the value of the "Last-Modified:" header with a randomly chosen time between the original value and the current time. In theory the server could send each document with a different "Last-Modified:" header to track visits without using cookies. "Randomize" makes it impossible and the browser can still revalidate cached documents.

"reset-to-request-time" overwrites the value of the "Last-Modified:" header with the current time. You could use this option together with `hided-if-modified-since` to further customize your random range.

The preferred parameter here is "randomize". It is safe to use, as long as the time settings are more or less correct. If the server sets the "Last-Modified:" header to the time of the request, the random range becomes zero and the value stays the same. Therefore you should later randomize it a second time with `hided-if-modified-since`, just to be sure.

It is also recommended to use this action together with `crunch-if-none-match`.

Example usage:

```
# Let the browser revalidate without being tracked across sessions
{ +hide-if-modified-since{-60} \
 +overwrite-last-modified{randomize} \
 +crunch-if-none-match}
/
```

8.5.31. redirect

Typical use:

Redirect requests to other sites.

Effect:

Convinces the browser that the requested document has been moved to another location and the browser should get it from there.

Type:

Parameterized

Parameter:

An absolute URL or a single pcrs command.

Notes:

Requests to which this action applies are answered with a HTTP redirect to URLs of your choosing. The new URL is either provided as parameter, or derived by applying a single pcrs command to the original URL.

This action will be ignored if you use it together with `block`. It can be combined with `fast-redirects{check-decoded-url}` to redirect to a decoded version of a rewritten URL.

Use this action carefully, make sure not to create redirection loops and be aware that using your own redirects might make it possible to fingerprint your requests.

Example usages:

```
# Replace example.com's style sheet with another one
{ +redirect{http://localhost/css-replacements/example.com.css} }
 example.com/stylesheet\.css

# Create a short, easy to remember nickname for a favorite site
# (relies on the browser accept and forward invalid URLs to Privoxy)
{ +redirect{http://www.privoxy.org/user-manual/actions-file.html} }
 a

# Always use the expanded view for Undeadly.org articles
# (Note the $ at the end of the URL pattern to make sure
# the request for the rewritten URL isn't redirected as well)
{+redirect{s@$@&mode=expanded@}}
undeadly.org/cgi\?action=article&sid=\d*$
```

8.5.32. send-vanilla-wafer

Typical use:

Feed log analysis scripts with useless data.

Effect:

Sends a cookie with each request stating that you do not accept any copyright on cookies sent to you, and asking the site operator not to track you.

Type:

Boolean.

Parameter:
N/A
Notes:
The vanilla wafer is a (relatively) unique header and could conceivably be used to track you.

This action is rarely used and not enabled in the default configuration.
Example usage:

```
+send-vanilla-wafer
```

8.5.33. send-wafer

Typical use:
Send custom cookies or feed log analysis scripts with even more useless data.
Effect:
Sends a custom, user-defined cookie with each request.
Type:
Multi-value.
Parameter:
A string of the form "*name=value*".
Notes:
Being multi-valued, multiple instances of this action can apply to the same request, resulting in multiple cookies being sent.

This action is rarely used and not enabled in the default configuration.
Example usage (section):

```
{+send-wafer{UsingPrivoxy=true}}
my-internal-testing-server.void
```

8.5.34. server-header-filter

Typical use:
Rewrite or remove single server headers.
Effect:
All server headers to which this action applies are filtered on-the-fly through the specified regular expression based substitutions.
Type:
Parameterized.
Parameter:
The name of a server-header filter, as defined in one of the filter files.
Notes:
Server-header filters are applied to each header on its own, not to all at once. This makes it easier to diagnose problems, but on the downside you can't write filters that only change header x if header y's value is z. You can do that by using tags though.

Server-header filters are executed after the other header actions have finished and use their output as input.

Please refer to the filter file chapter to learn which server-header filters are available by default, and how to create your own.
Example usage (section):

```
{+server-header-filter{html-to-xml}}
example.org/xml-instance-that-is-delivered-as-html

{+server-header-filter{xml-to-html}}
example.org/instance-that-is-delivered-as-xml-but-is-not
```

8.5.35. server-header-tagger

Typical use:
Enable or disable filters based on the Content-Type header.
Effect:
Server headers to which this action applies are filtered on-the-fly through the specified regular expression based substitutions, the result is used as tag.
Type:
Parameterized.
Parameter:
The name of a server-header tagger, as defined in one of the filter files.
Notes:
Server-header taggers are applied to each header on its own, and as the header isn't modified, each tagger "sees" the original.

Server-header taggers are executed before all other header actions that modify server headers. Their tags can be used to control all of the other server-header actions, the content filters and the crunch actions (redirect and block).

Obviously crunching based on tags created by server-header taggers doesn't prevent the request from showing up in the server's log file.

Example usage (section):

```
# Tag every request with the content type declared by the server
{+server-header-tagger{content-type}}
/
```

8.5.36. session-cookies-only

Typical use:

Allow only temporary "session" cookies (for the current browser session *only*).

Effect:

Deletes the "expires" field from "Set-Cookie:" server headers. Most browsers will not store such cookies permanently and forget them in between sessions.

Type:

Boolean.

Parameter:

N/A

Notes:

This is less strict than `crunch-incoming-cookies` / `crunch-outgoing-cookies` and allows you to browse websites that insist or rely on setting cookies, without compromising your privacy too badly.

Most browsers will not permanently store cookies that have been processed by `session-cookies-only` and will forget about them between sessions. This makes profiling cookies useless, but won't break sites which require cookies so that you can log in for transactions. This is generally turned on for all sites, and is the recommended setting.

It makes *no sense at all* to use `session-cookies-only` together with `crunch-incoming-cookies` or `crunch-outgoing-cookies`. If you do, cookies will be plainly killed.

Note that it is up to the browser how it handles such cookies without an "expires" field. If you use an exotic browser, you might want to try it out to be sure.

This setting also has no effect on cookies that may have been stored previously by the browser before starting Privoxy. These would have to be removed manually.

Privoxy also uses the content-cookies filter to block some types of cookies. Content cookies are not effected by `session-cookies-only`.

Example usage:

```
+session-cookies-only
```

8.5.37. set-image-blocker

Typical use:

Choose the replacement for blocked images

Effect:

This action alone doesn't do anything noticeable. If *both* `block` *and* `handle-as-image` *also* apply, i.e. if the request is to be blocked as an image, *then* the parameter of this action decides what will be sent as a replacement.

Type:

Parameterized.

Parameter:

- ◊ "pattern" to send a built-in checkerboard pattern image. The image is visually decent, scales very well, and makes it obvious where banners were busted.
- ◊ "blank" to send a built-in transparent image. This makes banners disappear completely, but makes it hard to detect where Privoxy has blocked images on a given page and complicates troubleshooting if Privoxy has blocked innocent images, like navigation icons.
- ◊ "*`target-url`*" to send a redirect to *`target-url`*. You can redirect to any image anywhere, even in your local filesystem via "file:///" URL. (But note that not all browsers support redirecting to a local file system).

  A good application of redirects is to use special Privoxy-built-in URLs, which send the built-in images, as *`target-url`*. This has the same visual effect as specifying "blank" or "pattern" in the first place, but enables your browser to cache the replacement image, instead of requesting it over and over again.

Notes:

The URLs for the built-in images are "http://config.privoxy.org/send-banner?type=*type*", where *type* is either "blank" or "pattern".

There is a third (advanced) type, called "auto". It is *NOT* to be used in `set-image-blocker`, but meant for use from filters. Auto will select the type of image that would have applied to the referring page, had it been an image.

Example usage:

Built-in pattern:

```
+set-image-blocker{pattern}
```

Redirect to the BSD daemon:

```
+set-image-blocker{http://www.freebsd.org/gifs/dae_up3.gif}
```

Redirect to the built-in pattern for better caching:

```
+set-image-blocker{http://config.privoxy.org/send-banner?type=pattern}
```

8.5.38. treat-forbidden-connects-like-blocks

Typical use:

Block forbidden connects with an easy to find error message.

Effect:

If this action is enabled, Privoxy no longer makes a difference between forbidden connects and ordinary blocks.

Type:

Boolean

Parameter:

N/A

Notes:

By default Privoxy answers forbidden "Connect" requests with a short error message inside the headers. If the browser doesn't display headers (most don't), you just see an empty page.

With this action enabled, Privoxy displays the message that is used for ordinary blocks instead. If you decide to make an exception for the page in question, you can do so by following the "See why" link.

For "Connect" requests the clients tell Privoxy which host they are interested in, but not which document they plan to get later. As a result, the "Go there anyway" wouldn't work and is therefore suppressed.

Example usage:

```
+treat-forbidden-connects-like-blocks
```

8.5.39. Summary

Note that many of these actions have the potential to cause a page to misbehave, possibly even not to display at all. There are many ways a site designer may choose to design his site, and what HTTP header content, and other criteria, he may depend on. There is no way to have hard and fast rules for all sites. See the Appendix for a brief example on troubleshooting actions.

8.6. Aliases

Custom "actions", known to Privoxy as "aliases", can be defined by combining other actions. These can in turn be invoked just like the built-in actions. Currently, an alias name can contain any character except space, tab, "=", "{" and "}", but we *strongly recommend* that you only use "a" to "z", "0" to "9", "+", and "-". Alias names are not case sensitive, and are not required to start with a "+" or "-" sign, since they are merely textually expanded.

Aliases can be used throughout the actions file, but they *must be defined in a special section at the top of the file!* And there can only be one such section per actions file. Each actions file may have its own alias section, and the aliases defined in it are only visible within that file.

There are two main reasons to use aliases: One is to save typing for frequently used combinations of actions, the other one is a gain in flexibility: If you decide once how you want to handle shops by defining an alias called "shop", you can later change your policy on shops in *one* place, and your changes will take effect everywhere in the actions file where the "shop" alias is used. Calling aliases by their purpose also makes your actions files more readable.

Currently, there is one big drawback to using aliases, though: Privoxy's built-in web-based action file editor honors aliases when reading the actions files, but it expands them before writing. So the effects of your aliases are of course preserved, but the aliases themselves are lost when you edit sections that use aliases with it.

Now let's define some aliases...

```
# Useful custom aliases we can use later.
```

```
#
# Note the (required!) section header line and that this section
# must be at the top of the actions file!
#
{{alias}}

# These aliases just save typing later:
# (Note that some already use other aliases!)
#
+crunch-all-cookies = +crunch-incoming-cookies +crunch-outgoing-cookies
-crunch-all-cookies = -crunch-incoming-cookies -crunch-outgoing-cookies
+block-as-image      = +block +handle-as-image
allow-all-cookies   = -crunch-all-cookies -session-cookies-only -filter{content-cookies}

# These aliases define combinations of actions
# that are useful for certain types of sites:
#
fragile     = -block -filter -crunch-all-cookies -fast-redirects -hide-referrer -kill-popups -prevent-compression

shop        = -crunch-all-cookies -filter{all-popups} -kill-popups

# Short names for other aliases, for really lazy people ;-)
#
c0 = +crunch-all-cookies
c1 = -crunch-all-cookies
```

...and put them to use. These sections would appear in the lower part of an actions file and define exceptions to the default actions (as specified further up for the "/" pattern):

```
# These sites are either very complex or very keen on
# user data and require minimal interference to work:
#
{fragile}
.office.microsoft.com
.windowsupdate.microsoft.com
# Gmail is really mail.google.com, not gmail.com
mail.google.com

# Shopping sites:
# Allow cookies (for setting and retrieving your customer data)
#
{shop}
.quietpc.com
.worldpay.com   # for quietpc.com
mybank.example.com

# These shops require pop-ups:
#
{-kill-popups -filter{all-popups} -filter{unsolicited-popups}}
 .dabs.com
 .overclockers.co.uk
```

Aliases like "shop" and "fragile" are typically used for "problem" sites that require more than one action to be disabled in order to function properly.

8.7. Actions Files Tutorial

The above chapters have shown which actions files there are and how they are organized, how actions are specified and applied to URLs, how patterns work, and how to define and use aliases. Now, let's look at an example `default.action` and `user.action` file and see how all these pieces come together:

8.7.1. default.action

Every config file should start with a short comment stating its purpose:

```
# Sample default.action file <ijbswa-developers@lists.sourceforge.net>
```

Then, since this is the `default.action` file, the first section is a special section for internal use that you needn't change or worry about:

```
############################################################################
# Settings -- Don't change! For internal Privoxy use ONLY.
############################################################################

{{settings}}
for-privoxy-version=3.0
```

After that comes the (optional) alias section. We'll use the example section from the above chapter on aliases, that also explains why and how aliases are used:

```
############################################################################
# Aliases
```

```
############################################################################
{{alias}}

 # These aliases just save typing later:
 # (Note that some already use other aliases!)
 #
 +crunch-all-cookies = +crunch-incoming-cookies +crunch-outgoing-cookies
 -crunch-all-cookies = -crunch-incoming-cookies -crunch-outgoing-cookies
 +block-as-image      = +block +handle-as-image
 mercy-for-cookies   = -crunch-all-cookies -session-cookies-only -filter{content-cookies}

 # These aliases define combinations of actions
 # that are useful for certain types of sites:
 #
 fragile     = -block -filter -crunch-all-cookies -fast-redirects -hide-referrer -kill-popups
 shop        = -crunch-all-cookies -filter{all-popups} -kill-popups
```

Now come the regular sections, i.e. sets of actions, accompanied by URL patterns to which they apply. Remember *all actions are disabled when matching starts*, so we have to explicitly enable the ones we want.

The first regular section is probably the most important. It has only one pattern, "`/`", but this pattern matches all URLs. Therefore, the set of actions used in this "default" section *will be applied to all requests as a start*. It can be partly or wholly overridden by later matches further down this file, or in user.action, but it will still be largely responsible for your overall browsing experience.

Again, at the start of matching, all actions are disabled, so there is no need to disable any actions here. (Remember: a "+" preceding the action name enables the action, a "-" disables!). Also note how this long line has been made more readable by splitting it into multiple lines with line continuation.

```
############################################################################
# "Defaults" section:
############################################################################
 { \
 +deanimate-gifs \
 +filter{html-annoyances} \
 +filter{refresh-tags} \
 +filter{webbugs} \
 +filter{ie-exploits} \
 +hide-forwarded-for-headers \
 +hide-from-header{block} \
 +hide-referrer{forge} \
 +prevent-compression \
 +session-cookies-only \
 +set-image-blocker{pattern} \
 }
 / # forward slash will match *all* potential URL patterns.
```

The default behavior is now set.

The first of our specialized sections is concerned with "fragile" sites, i.e. sites that require minimum interference, because they are either very complex or very keen on tracking you (and have mechanisms in place that make them unusable for people who avoid being tracked). We will simply use our pre-defined `fragile` alias instead of stating the list of actions explicitly:

```
############################################################################
# Exceptions for sites that'll break under the default action set:
############################################################################

# "Fragile" Use a minimum set of actions for these sites (see alias above):
#
{ fragile }
.office.microsoft.com           # surprise, surprise!
.windowsupdate.microsoft.com
mail.google.com
```

Shopping sites are not as fragile, but they typically require cookies to log in, and pop-up windows for shopping carts or item details. Again, we'll use a pre-defined alias:

```
# Shopping sites:
#
{ shop }
.quietpc.com
.worldpay.com   # for quietpc.com
.jungle.com
.scan.co.uk
```

The `fast-redirects` action, which we enabled per default above, breaks some sites. So disable it for popular sites where we know it misbehaves:

```
{ -fast-redirects }
login.yahoo.com
edit.*.yahoo.com
.google.com
.altavista.com/.*(like|url|link):http
```

```
.altavista.com/trans.*urltext=http
.nytimes.com
```

It is important that Privoxy knows which URLs belong to images, so that *if* they are to be blocked, a substitute image can be sent, rather than an HTML page. Contacting the remote site to find out is not an option, since it would destroy the loading time advantage of banner blocking, and it would feed the advertisers (in terms of money *and* information). We can mark any URL as an image with the `handle-as-image` action, and marking all URLs that end in a known image file extension is a good start:

```
############################################################################
# Images:
############################################################################

# Define which file types will be treated as images, in case they get
# blocked further down this file:
#
{ +handle-as-image }
/.*\.(gif|jpe?g|png|bmp|ico)$
```

And then there are known banner sources. They often use scripts to generate the banners, so it won't be visible from the URL that the request is for an image. Hence we block them *and* mark them as images in one go, with the help of our `+block-as-image` alias defined above. (We could of course just as well use `+block` `+handle-as-image` here.) Remember that the type of the replacement image is chosen by the `set-image-blocker` action. Since all URLs have matched the default section with its `+set-image-blocker{pattern}` action before, it still applies and needn't be repeated:

```
# Known ad generators:
#
{ +block-as-image }
ar.atwola.com
.ad.doubleclick.net
.ad.*.doubleclick.net
.a.yimg.com/(?:(?!/i/).)*$
.a[0-9].yimg.com/(?:(?!/i/).)*$
bs*.gsanet.com
.qkimg.net
```

One of the most important jobs of Privoxy is to block banners. Many of these can be "blocked" by the `filter{banners-by-size}` action, which we enabled above, and which deletes the references to banner images from the pages while they are loaded, so the browser doesn't request them anymore, and hence they don't need to be blocked here. But this naturally doesn't catch all banners, and some people choose not to use filters, so we need a comprehensive list of patterns for banner URLs here, and apply the `block` action to them.

First comes many generic patterns, which do most of the work, by matching typical domain and path name components of banners. Then comes a list of individual patterns for specific sites, which is omitted here to keep the example short:

```
############################################################################
# Block these fine banners:
############################################################################
{ +block }

# Generic patterns:
#
ad*.
.*ads.
banner?.
count*.
/.*count(er)?\.(pl|cgi|exe|dll|asp|php[34]?)
/(?:.*/)?(publicite|werbung|rekla(ma|me|am)|annonse|maino(kset|nta|s)?)/

# Site-specific patterns (abbreviated):
#
.hitbox.com
```

It's quite remarkable how many advertisers actually call their banner servers ads.*company*.com, or call the directory in which the banners are stored simply "banners". So the above generic patterns are surprisingly effective.

But being very generic, they necessarily also catch URLs that we don't want to block. The pattern `.*ads.` e.g. catches "nasty-*ads*.nasty-corp.com" as intended, but also "downlo*ads*.sourcefroge.net" or "*ads*l.some-provider.net." So here come some well-known exceptions to the `+block` section above.

Note that these are exceptions to exceptions from the default! Consider the URL "downloads.sourcefroge.net": Initially, all actions are deactivated, so it wouldn't get blocked. Then comes the defaults section, which matches the URL, but just deactivates the `block` action once again. Then it matches `.*ads.`, an exception to the general non-blocking policy, and suddenly `+block` applies. And now, it'll match `.*loads.`, where `-block` applies, so (unless it matches *again* further down) it ends up with no `block` action applying.

```
############################################################################
# Save some innocent victims of the above generic block patterns:
############################################################################

# By domain:
#
```

```
{ -block }
adv[io]*.  # (for advogato.org and advice.*)
adsl.      # (has nothing to do with ads)
adobe.     # (has nothing to do with ads either)
ad[ud]*.   # (adult.* and add.*)
.edu       # (universities don't host banners (yet!))
.*loads.   # (downloads, uploads etc)

# By path:
#
/.*loads/

# Site-specific:
#
www.globalintersec.com/adv # (adv = advanced)
www.ugu.com/sui/ugu/adv
```

Filtering source code can have nasty side effects, so make an exception for our friends at sourceforge.net, and all paths with "cvs" in them. Note that `-filter` disables *all* filters in one fell swoop!

```
# Don't filter code!
#
{ -filter }
/(.*/)?cvs
bugzilla.
developer.
wiki.
.sourceforge.net
```

The actual `default.action` is of course much more comprehensive, but we hope this example made clear how it works.

8.7.2. user.action

So far we are painting with a broad brush by setting general policies, which would be a reasonable starting point for many people. Now, you might want to be more specific and have customized rules that are more suitable to your personal habits and preferences. These would be for narrowly defined situations like your ISP or your bank, and should be placed in `user.action`, which is parsed after all other actions files and hence has the last word, over-riding any previously defined actions. `user.action` is also a *safe* place for your personal settings, since `default.action` is actively maintained by the Privoxy developers and you'll probably want to install updated versions from time to time.

So let's look at a few examples of things that one might typically do in `user.action`:

```
# My user.action file. <fred@example.com>
```

As aliases are local to the actions file that they are defined in, you can't use the ones from `default.action`, unless you repeat them here:

```
# Aliases are local to the file they are defined in.
# (Re-)define aliases for this file:
#
{{alias}}
#
# These aliases just save typing later, and the alias names should
# be self explanatory.
#
+crunch-all-cookies = +crunch-incoming-cookies +crunch-outgoing-cookies
-crunch-all-cookies = -crunch-incoming-cookies -crunch-outgoing-cookies
 allow-all-cookies  = -crunch-all-cookies -session-cookies-only
 allow-popups       = -filter{all-popups} -kill-popups
+block-as-image     = +block +handle-as-image
-block-as-image     = -block

# These aliases define combinations of actions that are useful for
# certain types of sites:
#
fragile     = -block -crunch-all-cookies -filter -fast-redirects -hide-referrer -kill-popups
shop        = -crunch-all-cookies allow-popups

# Allow ads for selected useful free sites:
#
allow-ads   = -block -filter{banners-by-size} -filter{banners-by-link}

# Alias for specific file types that are text, but might have conflicting
# MIME types. We want the browser to force these to be text documents.
handle-as-text = -filter +-content-type-overwrite{text/plain} +-force-text-mode -hide-content-disposition
```

Say you have accounts on some sites that you visit regularly, and you don't want to have to log in manually each time. So you'd like to allow persistent cookies for these sites. The `allow-all-cookies` alias defined above does exactly that, i.e. it disables crunching of cookies in any direction, and the processing of cookies to make them only temporary.

```
{ allow-all-cookies }
```

```
 sourceforge.net
 .yahoo.com
 .msdn.microsoft.com
 .redhat.com
```

Your bank is allergic to some filter, but you don't know which, so you disable them all:

```
{ -filter }
 .your-home-banking-site.com
```

Some file types you may not want to filter for various reasons:

```
# Technical documentation is likely to contain strings that might
# erroneously get altered by the JavaScript-oriented filters:
#
.tldp.org
/(.*/)?selfhtml/

# And this stupid host sends streaming video with a wrong MIME type,
# so that Privoxy thinks it is getting HTML and starts filtering:
#
stupid-server.example.com/
```

Example of a simple block action. Say you've seen an ad on your favourite page on example.com that you want to get rid of. You have right-clicked the image, selected "copy image location" and pasted the URL below while removing the leading http://, into a `{ +block }` section. Note that `{ +handle-as-image }` need not be specified, since all URLs ending in `.gif` will be tagged as images by the general rules as set in default.action anyway:

```
{ +block }
 www.example.com/nasty-ads/sponsor\.gif
 another.example.net/more/junk/here/
```

The URLs of dynamically generated banners, especially from large banner farms, often don't use the well-known image file name extensions, which makes it impossible for Privoxy to guess the file type just by looking at the URL. You can use the `+block-as-image` alias defined above for these cases. Note that objects which match this rule but then turn out NOT to be an image are typically rendered as a "broken image" icon by the browser. Use cautiously.

```
{ +block-as-image }
 .doubleclick.net
 .fastclick.net
 /Realmedia/ads/
 ar.atwola.com/
```

Now you noticed that the default configuration breaks Forbes Magazine, but you were too lazy to find out which action is the culprit, and you were again too lazy to give feedback, so you just used the `fragile` alias on the site, and -- *whoa!* -- it worked. The `fragile` aliases disables those actions that are most likely to break a site. Also, good for testing purposes to see if it is Privoxy that is causing the problem or not. We later find other regular sites that misbehave, and add those to our personalized list of troublemakers:

```
{ fragile }
 .forbes.com
 webmail.example.com
 .mybank.com
```

You like the "fun" text replacements in `default.filter`, but it is disabled in the distributed actions file. So you'd like to turn it on in your private, update-safe config, once and for all:

```
{ +filter{fun} }
 / # For ALL sites!
```

Note that the above is not really a good idea: There are exceptions to the filters in `default.action` for things that really shouldn't be filtered, like code on CVS->Web interfaces. Since `user.action` has the last word, these exceptions won't be valid for the "fun" filtering specified here.

You might also worry about how your favourite free websites are funded, and find that they rely on displaying banner advertisements to survive. So you might want to specifically allow banners for those sites that you feel provide value to you:

```
{ allow-ads }
 .sourceforge.net
 .slashdot.org
 .osdn.net
```

Note that `allow-ads` has been aliased to `-block`, `-filter{banners-by-size}`, and `-filter{banners-by-link}` above.

Invoke another alias here to force an over-ride of the MIME type `application/x-sh` which typically would open a download type dialog. In my case, I want to look at the shell script, and then I can save it should I choose to.

```
{ handle-as-text }
 /.*\.sh$
```

`user.action` is generally the best place to define exceptions and additions to the default policies of `default.action`. Some actions are safe to have their default policies set here though. So let's set a default policy to have a "blank" image as opposed to the checkerboard pattern for *ALL* sites. "/" of course matches all URL paths and patterns:

```
{ +set-image-blocker{blank} }
/ # ALL sites
```

9. Filter Files

On-the-fly text substitutions need to be defined in a "filter file". Once defined, they can then be invoked as an "action".

Privoxy supports three different filter actions: `filter` to rewrite the content that is send to the client, `client-header-filter` to rewrite headers that are send by the client, and `server-header-filter` to rewrite headers that are send by the server.

Privoxy also supports two tagger actions: `client-header-tagger` and `server-header-tagger`. Taggers and filters use the same syntax in the filter files, the difference is that taggers don't modify the text they are filtering, but use a rewritten version of the filtered text as tag. The tags can then be used to change the applying actions through sections with tag-patterns.

Multiple filter files can be defined through the `filterfile` config directive. The filters as supplied by the developers are located in `default.filter`. It is recommended that any locally defined or modified filters go in a separately defined file such as `user.filter`.

Common tasks for content filters are to eliminate common annoyances in HTML and JavaScript, such as pop-up windows, exit consoles, crippled windows without navigation tools, the infamous <BLINK> tag etc, to suppress images with certain width and height attributes (standard banner sizes or web-bugs), or just to have fun.

Enabled content filters are applied to any content whose "Content Type" header is recognised as a sign of text-based content, with the exception of `text/plain`. Use the force-text-mode action to also filter other content.

Substitutions are made at the source level, so if you want to "roll your own" filters, you should first be familiar with HTML syntax, and, of course, regular expressions.

Just like the actions files, the filter file is organized in sections, which are called *filters* here. Each filter consists of a heading line, that starts with one of the *keywords* `FILTER:`, `CLIENT-HEADER-FILTER:` or `SERVER-HEADER-FILTER:` followed by the filter's *name*, and a short (one line) *description* of what it does. Below that line come the *jobs*, i.e. lines that define the actual text substitutions. By convention, the name of a filter should describe what the filter *eliminates*. The comment is used in the web-based user interface.

Once a filter called *name* has been defined in the filter file, it can be invoked by using an action of the form +`filter`{*name*} in any actions file.

Filter definitions start with a header line that contains the filter type, the filter name and the filter description. A content filter header line for a filter called "foo" could look like this:

```
FILTER: foo Replace all "foo" with "bar"
```

Below that line, and up to the next header line, come the jobs that define what text replacements the filter executes. They are specified in a syntax that imitates Perl's `s///` operator. If you are familiar with Perl, you will find this to be quite intuitive, and may want to look at the PCRS documentation for the subtle differences to Perl behaviour. Most notably, the non-standard option letter `U` is supported, which turns the default to ungreedy matching.

If you are new to "Regular Expressions", you might want to take a look at the Appendix on regular expressions, and see the Perl manual for the `s///` operator's syntax and Perl-style regular expressions in general. The below examples might also help to get you started.

9.1. Filter File Tutorial

Now, let's complete our "foo" content filter. We have already defined the heading, but the jobs are still missing. Since all it does is to replace "foo" with "bar", there is only one (trivial) job needed:

```
s/foo/bar/
```

But wait! Didn't the comment say that *all* occurrences of "foo" should be replaced? Our current job will only take care of the first "foo" on each page. For global substitution, we'll need to add the `g` option:

```
s/foo/bar/g
```

Our complete filter now looks like this:

```
FILTER: foo Replace all "foo" with "bar"
s/foo/bar/g
```

Let's look at some real filters for more interesting examples. Here you see a filter that protects against some common annoyances that arise from JavaScript abuse. Let's look at its jobs one after the other:

```
FILTER: js-annoyances Get rid of particularly annoying JavaScript abuse

# Get rid of JavaScript referrer tracking. Test page: http://www.randomoddness.com/untitled.htm
#
s|(<script.*)document\.referrer(.*</script>)|$1"Not Your Business!"$2|Usg
```

Following the header line and a comment, you see the job. Note that it uses `|` as the delimiter instead of `/`, because the pattern contains a forward slash, which would otherwise have to be escaped by a backslash (`\`).

Now, let's examine the pattern: it starts with the text `<script.*` enclosed in parentheses. Since the dot matches any character, and * means: "Match an arbitrary number of the element left of myself", this matches "<script", followed by *any* text, i.e. it matches the whole page, from the start of the first <script> tag.

That's more than we want, but the pattern continues: `document\.referrer` matches only the exact string "document.referrer". The dot needed to be *escaped*, i.e. preceded by a backslash, to take away its special meaning as a joker, and make it just a regular dot. So far, the meaning is: Match from the start of the first <script> tag in a the page, up to, and including, the text "document.referrer", if *both* are present in the page (and appear in that order).

But there's still more pattern to go. The next element, again enclosed in parentheses, is `.*</script>`. You already know what `.*` means, so the whole pattern translates to: Match from the start of the first <script> tag in a page to the end of the last <script> tag, provided that the text "document.referrer" appears somewhere in between.

This is still not the whole story, since we have ignored the options and the parentheses: The portions of the page matched by sub-patterns that are enclosed in parentheses, will be remembered and be available through the variables `$1, $2, ...` in the substitute. The `U` option switches to ungreedy matching, which means that the first `.*` in the pattern will only "eat up" all text in between "<script" and the *first* occurrence of "document.referrer", and that the second `.*` will only span the text up to the *first* "</script>" tag. Furthermore, the `s` option says that the match may span multiple lines in the page, and the `g` option again means that the substitution is global.

So, to summarize, the pattern means: Match all scripts that contain the text "document.referrer". Remember the parts of the script from (and including) the start tag up to (and excluding) the string "document.referrer" as `$1`, and the part following that string, up to and including the closing tag, as `$2`.

Now the pattern is deciphered, but wasn't this about substituting things? So lets look at the substitute: `$1"Not Your Business!"$2` is easy to read: The text remembered as `$1`, followed by `"Not Your Business!"` (*including* the quotation marks!), followed by the text remembered as `$2`. This produces an exact copy of the original string, with the middle part (the "document.referrer") replaced by `"Not Your Business!"`.

The whole job now reads: Replace "document.referrer" by `"Not Your Business!"` wherever it appears inside a <script> tag. Note that this job won't break JavaScript syntax, since both the original and the replacement are syntactically valid string objects. The script just won't have access to the referrer information anymore.

We'll show you two other jobs from the JavaScript taming department, but this time only point out the constructs of special interest:

```
# The status bar is for displaying link targets, not pointless blahblah
#
s/window\.status\s*=\s*(['"]).*?\1/dUmMy=1/ig
```

`\s` stands for whitespace characters (space, tab, newline, carriage return, form feed), so that `\s*` means: "zero or more whitespace". The `?` in `.*?` makes this matching of arbitrary text ungreedy. (Note that the `U` option is not set). The `['"]` construct means: "a single *or* a double quote". Finally, `\1` is a back-reference to the first parenthesis just like `$1` above, with the difference that in the *pattern*, a backslash indicates a back-reference, whereas in the *substitute*, it's the dollar.

So what does this job do? It replaces assignments of single- or double-quoted strings to the "window.status" object with a dummy assignment (using a variable name that is hopefully odd enough not to conflict with real variables in scripts). Thus, it catches many cases where e.g. pointless descriptions are displayed in the status bar instead of the link target when you move your mouse over links.

```
# Kill OnUnload popups. Yummy. Test: http://www.zdnet.com/zdsubs/yahoo/tree/yfs.html
#
s/(<body [^>]*)onunload(.*>)/$1never$2/iU
```

Including the OnUnload event binding in the HTML DOM was a *CRIME*. When I close a browser window, I want it to close and die. Basta. This job replaces the "onunload" attribute in "<body>" tags with the dummy word `never`. Note that the `i` option makes the pattern matching case-insensitive. Also note that ungreedy matching alone doesn't always guarantee a minimal match: In the first parenthesis, we had to use `[^>]*` instead of `.*` to prevent the match from exceeding the <body> tag if it doesn't contain "OnUnload", but the page's content does.

The last example is from the fun department:

```
FILTER: fun Fun text replacements

# Spice the daily news:
#
s/microsoft(?!\.com)/MicroSuck/ig
```

Note the `(?!\.com)` part (a so-called negative lookahead) in the job's pattern, which means: Don't match, if the string ".com" appears directly following "microsoft" in the page. This prevents links to microsoft.com from being trashed, while still replacing the word everywhere else.

```
# Buzzword Bingo (example for extended regex syntax)
#
s* industry[ -]leading \
|  cutting[ -]edge \
|  customer[ -]focused \
```

```
|  market[ -]driven \
|  award[ -]winning # Comments are OK, too! \
|  high[ -]performance \
|  solutions[ -]based \
|  unmatched \
|  unparalleled \
|  unrivalled \
*<font color="red"><b>BINGO!</b></font> \
*igx
```

The x option in this job turns on extended syntax, and allows for e.g. the liberal use of (non-interpreted!) whitespace for nicer formatting.

You get the idea?

9.2. The Pre-defined Filters

The distribution `default.filter` file contains a selection of pre-defined filters for your convenience:

js-annoyances
: The purpose of this filter is to get rid of particularly annoying JavaScript abuse. To that end, it

  ◊ replaces JavaScript references to the browser's referrer information with the string "Not Your Business!". This compliments the `hide-referrer` action on the content level.
  ◊ removes the bindings to the DOM's unload event which we feel has no right to exist and is responsible for most "exit consoles", i.e. nasty windows that pop up when you close another one.
  ◊ removes code that causes new windows to be opened with undesired properties, such as being full-screen, non-resizeable, without location, status or menu bar etc.

  Use with caution. This is an aggressive filter, and can break sites that rely heavily on JavaScript.

js-events
: This is a very radical measure. It removes virtually all JavaScript event bindings, which means that scripts can not react to user actions such as mouse movements or clicks, window resizing etc, anymore. Use with caution!

  We *strongly discourage* using this filter as a default since it breaks many legitimate scripts. It is meant for use only on extra-nasty sites (should you really need to go there).

html-annoyances
: This filter will undo many common instances of HTML based abuse.

  The `BLINK` and `MARQUEE` tags are neutralized (yeah baby!), and browser windows will be created as resizeable (as of course they should be!), and will have location, scroll and menu bars -- even if specified otherwise.

content-cookies
: Most cookies are set in the HTTP dialog, where they can be intercepted by the `crunch-incoming-cookies` and `crunch-outgoing-cookies` actions. But web sites increasingly make use of HTML meta tags and JavaScript to sneak cookies to the browser on the content level.

  This filter disables most HTML and JavaScript code that reads or sets cookies. It cannot detect all clever uses of these types of code, so it should not be relied on as an absolute fix. Use it wherever you would also use the cookie crunch actions.

refresh tags
: Disable any refresh tags if the interval is greater than nine seconds (so that redirections done via refresh tags are not destroyed). This is useful for dial-on-demand setups, or for those who find this HTML feature annoying.

unsolicited-popups
: This filter attempts to prevent only "unsolicited" pop-up windows from opening, yet still allow pop-up windows that the user has explicitly chosen to open. It was added in version 3.0.1, as an improvement over earlier such filters.

  Technical note: The filter works by redefining the window.open JavaScript function to a dummy function, `PrivoxyWindowOpen()`, during the loading and rendering phase of each HTML page access, and restoring the function afterward.

  This is recommended only for browsers that cannot perform this function reliably themselves. And be aware that some sites require such windows in order to function normally. Use with caution.

all-popups
: Attempt to prevent *all* pop-up windows from opening. Note this should be used with even more discretion than the above, since it is more likely to break some sites that require pop-ups for normal usage. Use with caution.

img-reorder
: This is a helper filter that has no value if used alone. It makes the `banners-by-size` and `banners-by-link` (see below) filters more effective and should be enabled together with them.

banners-by-size
: This filter removes image tags purely based on what size they are. Fortunately for us, many ads and banner images tend to conform to certain standardized sizes, which makes this filter quite effective for ad stripping purposes.

  Occasionally this filter will cause false positives on images that are not ads, but just happen to be of one of the standard banner sizes.

Recommended only for those who require extreme ad blocking. The default block rules should catch 95+% of all ads *without* this filter enabled.

banners-by-link
: This is an experimental filter that attempts to kill any banners if their URLs seem to point to known or suspected click trackers. It is currently not of much value and is not recommended for use by default.

webbugs
: Webbugs are small, invisible images (technically 1X1 GIF images), that are used to track users across websites, and collect information on them. As an HTML page is loaded by the browser, an embedded image tag causes the browser to contact a third-party site, disclosing the tracking information through the requested URL and/or cookies for that third-party domain, without the user ever becoming aware of the interaction with the third-party site. HTML-ized spam also uses a similar technique to verify email addresses.

  This filter removes the HTML code that loads such "webbugs".

tiny-textforms
: A rather special-purpose filter that can be used to enlarge textareas (those multi-line text boxes in web forms) and turn off hard word wrap in them. It was written for the sourceforge.net tracker system where such boxes are a nuisance, but it can be handy on other sites, too.

  It is not recommended to use this filter as a default.

jumping-windows
: Many consider windows that move, or resize themselves to be abusive. This filter neutralizes the related JavaScript code. Note that some sites might not display or behave as intended when using this filter. Use with caution.

frameset-borders
: Some web designers seem to assume that everyone in the world will view their web sites using the same browser brand and version, screen resolution etc, because only that assumption could explain why they'd use static frame sizes, yet prevent their frames from being resized by the user, should they be too small to show their whole content.

  This filter removes the related HTML code. It should only be applied to sites which need it.

demoronizer
: Many Microsoft products that generate HTML use non-standard extensions (read: violations) of the ISO 8859-1 aka Latin-1 character set. This can cause those HTML documents to display with errors on standard-compliant platforms.

  This filter translates the MS-only characters into Latin-1 equivalents. It is not necessary when using MS products, and will cause corruption of all documents that use 8-bit character sets other than Latin-1. It's mostly worthwhile for Europeans on non-MS platforms, if weird garbage characters sometimes appear on some pages, or user agents that don't correct for this on the fly.

shockwave-flash
: A filter for shockwave haters. As the name suggests, this filter strips code out of web pages that is used to embed shockwave flash objects.

quicktime-kioskmode
: Change HTML code that embeds Quicktime objects so that kioskmode, which prevents saving, is disabled.

fun
: Text replacements for subversive browsing fun. Make fun of your favorite Monopolist or play buzzword bingo.

crude-parental
: A demonstration-only filter that shows how Privoxy can be used to delete web content on a keyword basis.

ie-exploits
: An experimental collection of text replacements to disable malicious HTML and JavaScript code that exploits known security holes in Internet Explorer.

  Presently, it only protects against Nimda and a cross-site scripting bug, and would need active maintenance to provide more substantial protection.

site-specifics
: Some web sites have very specific problems, the cure for which doesn't apply anywhere else, or could even cause damage on other sites.

  This is a collection of such site-specific cures which should only be applied to the sites they were intended for, which is what the supplied `default.action` file does. Users shouldn't need to change anything regarding this filter.

google
: A CSS based block for Google text ads. Also removes a width limitation and the toolbar advertisement.

yahoo
: Another CSS based block, this time for Yahoo text ads. And removes a width limitation as well.

msn
: Another CSS based block, this time for MSN text ads. And removes tracking URLs, as well as a width limitation.

blogspot
: Cleans up some Blogspot blogs. Read the fine print before using this one!

  This filter also intentionally removes some navigation stuff and sets the page width to 100%. As a result, some rounded "corners" would appear to early or not at all and as fixing this would require a browser that understands background-size (CSS3), they are removed instead.

xml-to-html

Server-header filter to change the Content-Type from xml to html.

html-to-xml

Server-header filter to change the Content-Type from html to xml.

no-ping

Removes the non-standard `ping` attribute from anchor and area HTML tags.

hide-tor-exit-notation

Client-header filter to remove the **Tor** exit node notation found in Host and Referer headers.

If Privoxy and **Tor** are chained and Privoxy is configured to use socks4a, one can use "http://www.example.org.foobar.exit/" to access the host "www.example.org" through the **Tor** exit node "foobar".

As the HTTP client isn't aware of this notation, it treats the whole string "www.example.org.foobar.exit" as host and uses it for the "Host" and "Referer" headers. From the server's point of view the resulting headers are invalid and can cause problems.

An invalid "Referer" header can trigger "hot-linking" protections, an invalid "Host" header will make it impossible for the server to find the right vhost (several domains hosted on the same IP address).

This client-header filter removes the "foo.exit" part in those headers to prevent the mentioned problems. Note that it only modifies the HTTP headers, it doesn't make it impossible for the server to detect your **Tor** exit node based on the IP address the request is coming from.

---

10. Privoxy's Template Files

All Privoxy built-in pages, i.e. error pages such as the "404 - No Such Domain" error page, the "BLOCKED" page and all pages of its web-based user interface, are generated from *templates*. (Privoxy must be running for the above links to work as intended.)

These templates are stored in a subdirectory of the configuration directory called `templates`. On Unixish platforms, this is typically `/etc/privoxy/templates/`.

The templates are basically normal HTML files, but with place-holders (called symbols or exports), which Privoxy fills at run time. It is possible to edit the templates with a normal text editor, should you want to customize them. (*Not recommended for the casual user*). Should you create your own custom templates, you should use the `config` setting templdir to specify an alternate location, so your templates do not get overwritten during upgrades.

Note that just like in configuration files, lines starting with # are ignored when the templates are filled in.

The place-holders are of the form `@name@`, and you will find a list of available symbols, which vary from template to template, in the comments at the start of each file. Note that these comments are not always accurate, and that it's probably best to look at the existing HTML code to find out which symbols are supported and what they are filled in with.

A special application of this substitution mechanism is to make whole blocks of HTML code disappear when a specific symbol is set. We use this for many purposes, one of them being to include the beta warning in all our user interface (CGI) pages when Privoxy is in an alpha or beta development stage:

```
<!-- @if-unstable-start -->

  ... beta warning HTML code goes here ...

<!-- if-unstable-end@ -->
```

If the "unstable" symbol is set, everything in between and including `@if-unstable-start` and `if-unstable-end@` will disappear, leaving nothing but an empty comment:

```
<!--  -->
```

There's also an if-then-else construct and an `#include` mechanism, but you'll sure find out if you are inclined to edit the templates ;-)

All templates refer to a style located at `http://config.privoxy.org/send-stylesheet`. This is, of course, locally served by Privoxy and the source for it can be found and edited in the `cgi-style.css` template.

11. Contacting the Developers, Bug Reporting and Feature Requests

We value your feedback. In fact, we rely on it to improve Privoxy and its configuration. However, please note the following hints, so we can provide you with the best support:

11.1. Get Support

For casual users, our support forum at SourceForge is probably best suited: http://sourceforge.net/tracker/?group_id=11118&atid=211118

All users are of course welcome to discuss their issues on the users mailing list, where the developers also hang around.

Note that the Privoxy mailing lists are moderated. Posts from unsubscribed addresses have to be accepted manually by a moderator. This may cause a delay of several days and if you use a subject that doesn't clearly mention Privoxy or one of its features, your message may be accidentally discarded as spam.

If you aren't subscribed, you should therefore spend a few seconds to come up with a proper subject. Additionally you should make it clear that you want to get CC'd. Otherwise some responses will be directed to the mailing list only, and you won't see them.

11.2. Reporting Problems

"Problems" for our purposes, come in two forms:

- Configuration issues, such as ads that slip through, or sites that don't function properly due to one Privoxy "action" or another being turned "on".
- "Bugs" in the programming code that makes up Privoxy, such as that might cause a crash.

11.2.1. Reporting Ads or Other Configuration Problems

Please send feedback on ads that slipped through, innocent images that were blocked, sites that don't work properly, and other configuration related problem of `default.action` file, to http://sourceforge.net/tracker/?group_id=11118&atid=460288, the Actions File Tracker.

New, improved `default.action` files may occasionally be made available based on your feedback. These will be announced on the ijbswa-announce list and available from our the files section of our project page.

11.2.2. Reporting Bugs

Please report all bugs through our bug tracker: http://sourceforge.net/tracker/?group_id=11118&atid=111118.

Before doing so, please make sure that the bug has *not already been submitted* and observe the additional hints at the top of the submit form. If already submitted, please feel free to add any info to the original report that might help to solve the issue.

Please try to verify that it is a Privoxy bug, and not a browser or site bug or documented behaviour that just happens to be different than what you expected. If unsure, try toggling off Privoxy, and see if the problem persists.

If you are using your own custom configuration, please try the stock configs to see if the problem is configuration related. If you're having problems with a feature that is disabled by default, please ask around on the mailing list if others can reproduce the problem.

If you aren't using the latest Privoxy version, the bug may have been found and fixed in the meantime. We would appreciate if you could take the time to upgrade to the latest version (or even the latest CVS snapshot) and verify that your bug still exists.

Please be sure to provide the following information:

- The exact Privoxy version you are using (if you got the source from CVS, please also provide the source code revisions as shown in http://config.privoxy.org/show-version).
- The operating system and versions you run Privoxy on, (e.g. Windows XP SP2), if you are using a Unix flavor, sending the output of "uname -a" should do, in case of GNU/Linux, please also name the distribution.
- The name, platform, and version of the browser you were using (e.g. Internet Explorer v5.5 for Mac).
- The URL where the problem occurred, or some way for us to duplicate the problem (e.g. `http://somesite.example.com/?somethingelse=123`).
- Whether your version of Privoxy is one supplied by the Privoxy developers via SourceForge, or if you got your copy somewhere else.
- Whether you are using Privoxy in tandem with another proxy such as Tor. If so, please temporary disable the other proxy to see if the symptoms change.
- Whether you are using a personal firewall product. If so, does Privoxy work without it?
- Any other pertinent information to help identify the problem such as config or log file excerpts (yes, you should have log file entries for each action taken).

You don't have to tell us your actual name when filing a problem report, but please use a nickname so we can differentiate between your messages and the ones entered by other "anonymous" users that may respond to your request if they have the same problem or already found a solution.

Please also check the status of your request a few days after submitting it, as we may request additional information. If you use a SF id, you should automatically get a mail when someone responds to your request.

The appendix of the Privoxy User Manual also has helpful information on understanding `actions`, and `action` debugging.

11.3. Request New Features

You are welcome to submit ideas on new features or other proposals for improvement through our feature request tracker at http://sourceforge.net/tracker/?atid=361118&group_id=11118.

11.4. Other

For any other issues, feel free to use the mailing lists. Technically interested users and people who wish to contribute to the project are also welcome on the developers list! You can find an overview of all Privoxy-related mailing lists, including list archives, at: http://sourceforge.net/mail/?group_id=11118.

12. Privoxy Copyright, License and History

12.1. License

12.2. History

A long time ago, there was the Internet Junkbuster, by Anonymous Coders and Junkbusters Corporation. This saved many users a lot of pain in the early days of web advertising and user tracking.

But the web, its protocols and standards, and with it, the techniques for forcing ads on users, give up autonomy over their browsing, and for tracking them, keeps evolving. Unfortunately, the Internet Junkbuster did not. Version 2.0.2, published in 1998, was (and is) the last official release available from Junkbusters Corporation. Fortunately, it had been released under the GNU GPL, which allowed further development by others.

So Stefan Waldherr started maintaining an improved version of the software, to which eventually a number of people contributed patches. It could already replace banners with a transparent image, and had a first version of pop-up killing, but it was still very closely based on the original, with all its limitations, such as the lack of HTTP/1.1 support, flexible per-site configuration, or content modification. The last release from this effort was version 2.0.2-10, published in 2000.

Then, some developers picked up the thread, and started turning the software inside out, upside down, and then reassembled it, adding many new features along the way.

The result of this is Privoxy, whose first stable version, 3.0, was released August, 2002.

12.3. Authors

Current Privoxy Team:

```
Fabian Keil, lead developer
David Schmidt, developer

Hal Burgiss
Gerry Murphy
Roland Rosenfeld
Jörg Strohmayer
```

Former Privoxy Team Members:

```
Johny Agotnes
Rodrigo Barbosa
Moritz Barsnick
Ian Cummings
Brian Dessent
Jon Foster
Karsten Hopp
Alexander Lazic
Daniel Leite
Gábor Lipták
```

```
Adam Lock
Guy Laroche
Mark Martinec
Justin McMurtry
Andreas Oesterhelt
Haroon Rafique
Georg Sauthoff
Thomas Steudten
Rodney Stromlund
Sviatoslav Sviridov
Sarantis Paskalis
Stefan Waldherr
```

Thanks to the many people who have tested Privoxy, reported bugs, provided patches, made suggestions or contributed in some way. These include (in alphabetical order):

```
Ken Arromdee
Devin Bayer
Gergely Bor
Reiner Buehl
Andrew J. Caines
Clifford Caoile
Frédéric Crozat
Michael T. Davis
Mattes Dolak
Peter E.
Florian Effenberger
Markus Elfring
Dean Gaudet
Stephen Gildea
Daniel Griscom
Felix Gröbert
Aaron Hamid
Darel Henman
Magnus Holmgren
Ralf Horstmann
Stefan Huehner
Peter Hyman
Derek Jennings
Petr Kadlec
David Laight
Bert van Leeuwen
Don Libes
Paul Lieverse
Toby Lyward
Wil Mahan
Jindrich Makovicka
David Mediavilla
Raphael Moll
Amuro Namie
Adam Piggott
Dan Price
Lee R.
Roberto Ragusa
Félix Rauch
Maynard Riley
Chung-chieh Shan
Spinor S.
Bart Schelstraete
Oliver Stoeneberg
Peter Thoenen
Martin Thomas
Song Weijia
Jörg Weinmann
Darren Wiebe
Bobby G. Vinyard
Anduin Withers
Oliver Yeoh
Jamie Zawinski
```

Privoxy is based in part on code originally developed by Junkbusters Corp. and Anonymous Coders.

Privoxy heavily relies on Philip Hazel's PCRE.

The code to filter compressed content makes use of zlib which is written by Jean-loup Gailly and Mark Adler.

On systems that lack snprintf(), Privoxy is using a version written by Mark Martinec. On systems that lack strptime(), Privoxy is using the one from the GNU C Library written by Ulrich Drepper.

---

13. See Also

Other references and sites of interest to Privoxy users:

http://www.privoxy.org/, the Privoxy Home page.

http://www.privoxy.org/faq/, the Privoxy FAQ.

http://sourceforge.net/projects/ijbswa/, the Project Page for Privoxy on SourceForge.

http://config.privoxy.org/, the web-based user interface. Privoxy must be running for this to work. Shortcut: http://p.p/

http://sourceforge.net/tracker/?group_id=11118&atid=460288, to submit "misses" and other configuration related suggestions to the developers.

http://www.junkbusters.com/ht/en/cookies.html, an explanation how cookies are used to track web users.

http://www.junkbusters.com/ijb.html, the original Internet Junkbuster.

http://privacy.net/, a useful site to check what information about you is leaked while you browse the web.

http://www.squid-cache.org/, a popular caching proxy, which is often used together with Privoxy.

http://www.pps.jussieu.fr/~jch/software/polipo/, Polipo is a caching proxy with advanced features like pipelining, multiplexing and caching of partial instances. In many setups it can be used as Squid replacement.

http://tor.eff.org/, Tor can help anonymize web browsing, web publishing, instant messaging, IRC, SSH, and other applications.

http://www.privoxy.org/developer-manual/, the Privoxy developer manual.

14. Appendix

14.1. Regular Expressions

Privoxy uses Perl-style "regular expressions" in its actions files and filter file, through the PCRE and PCRS libraries.

If you are reading this, you probably don't understand what "regular expressions" are, or what they can do. So this will be a very brief introduction only. A full explanation would require a book ;-)

Regular expressions provide a language to describe patterns that can be run against strings of characters (letter, numbers, etc), to see if they match the string or not. The patterns are themselves (sometimes complex) strings of literal characters, combined with wild-cards, and other special characters, called meta-characters. The "meta-characters" have special meanings and are used to build complex patterns to be matched against. Perl Compatible Regular Expressions are an especially convenient "dialect" of the regular expression language.

To make a simple analogy, we do something similar when we use wild-card characters when listing files with the **dir** command in DOS. `*.*` matches all filenames. The "special" character here is the asterisk which matches any and all characters. We can be more specific and use ? to match just individual characters. So "dir file?.text" would match "file1.txt", "file2.txt", etc. We are pattern matching, using a similar technique to "regular expressions"!

Regular expressions do essentially the same thing, but are much, much more powerful. There are many more "special characters" and ways of building complex patterns however. Let's look at a few of the common ones, and then some examples:

. - Matches any single character, e.g. "a", "A", "4", ":", or "@".

? - The preceding character or expression is matched ZERO or ONE times. Either/or.

+ - The preceding character or expression is matched ONE or MORE times.

* - The preceding character or expression is matched ZERO or MORE times.

\ - The "escape" character denotes that the following character should be taken literally. This is used where one of the special characters (e.g. ".") needs to be taken literally and not as a special meta-character. Example: "example\.com", makes sure the period is recognized only as a period (and not expanded to its meta-character meaning of any single character).

[] - Characters enclosed in brackets will be matched if any of the enclosed characters are encountered. For instance, "[0-9]" matches any numeric digit (zero through nine). As an example, we can combine this with "+" to match any digit one of more times: "[0-9]+".

() - parentheses are used to group a sub-expression, or multiple sub-expressions.

| - The "bar" character works like an "or" conditional statement. A match is successful if the sub-expression on either side of "|" matches. As an example: "/(this|that) example/" uses grouping and the bar character and would match either "this example" or "that example", and nothing else.

These are just some of the ones you are likely to use when matching URLs with Privoxy, and is a long way from a definitive list. This is enough to get us started with a few simple examples which may be more illuminating:

`/.*/banners/.*` - A simple example that uses the common combination of "." and "*" to denote any character, zero or more times. In other words, any string at all. So we start with a literal forward slash, then our regular expression pattern (".*") another literal forward slash, the string "banners", another forward slash, and lastly another ".*". We are building a directory path here. This will match any file with the path that has a directory named "banners" in it. The ".*" matches any characters, and this could conceivably be more forward slashes, so it might expand into a much longer looking path. For example, this could match: "/eye/hate/spammers/banners/annoy_me_please.gif", or just "/banners/annoying.html", or almost an infinite number of other possible combinations, just so it has "banners" in the path somewhere.

And now something a little more complex:

`/.*/adv((er)?ts?|ertis(ing|ements?))?/` - We have several literal forward slashes again ("/"), so we are building another expression that is a file path statement. We have another ".*", so we are matching against any conceivable sub-path, just so it matches our expression. The only true literal that *must match* our pattern is adv, together with the forward slashes. What comes after the "adv" string is the interesting part.

Remember the "?" means the preceding expression (either a literal character or anything grouped with "(...)" in this case) can exist or not, since this means either zero or one match. So "((er)?ts?|ertis(ing|ements?))" is optional, as are the individual sub-expressions: "(er)", "(ing|ements?)", and the "s". The "|" means "or". We have two of those. For instance, "(ing|ements?)", can expand to match either "ing" *OR* "ements?". What is being done here, is an attempt at matching as many variations of "advertisement", and similar, as possible. So

this would expand to match just "adv", or "advert", or "adverts", or "advertising", or "advertisement", or "advertisements". You get the idea. But it would not match "advertizements" (with a "z"). We could fix that by changing our regular expression to: "/.*/adv((er)?ts?|erti(s|z)(ing|ements?))?/", which would then match either spelling.

`/.*/advert[0-9]+\.(gif|jpe?g)` - Again another path statement with forward slashes. Anything in the square brackets "[ ]" can be matched. This is using "0-9" as a shorthand expression to mean any digit one through nine. It is the same as saying "0123456789". So any digit matches. The "+" means one or more of the preceding expression must be included. The preceding expression here is what is in the square brackets -- in this case, any digit one through nine. Then, at the end, we have a grouping: "(gif|jpe?g)". This includes a "|", so this needs to match the expression on either side of that bar character also. A simple "gif" on one side, and the other side will in turn match either "jpeg" or "jpg", since the "?" means the letter "e" is optional and can be matched once or not at all. So we are building an expression here to match image GIF or JPEG type image file. It must include the literal string "advert", then one or more digits, and a "." (which is now a literal, and not a special character, since it is escaped with "\"), and lastly either "gif", or "jpeg", or "jpg". Some possible matches would include: "//advert1.jpg", "/nasty/ads/advert1234.gif", "/banners/from/hell/advert99.jpg". It would not match "advert1.gif" (no leading slash), or "/adverts232.jpg" (the expression does not include an "s"), or "/advert1.jsp" ("jsp" is not in the expression anywhere).

We are barely scratching the surface of regular expressions here so that you can understand the default Privoxy configuration files, and maybe use this knowledge to customize your own installation. There is much, much more that can be done with regular expressions. Now that you know enough to get started, you can learn more on your own :/

More reading on Perl Compatible Regular expressions: http://perldoc.perl.org/perlre.html

For information on regular expression based substitutions and their applications in filters, please see the filter file tutorial in this manual.

14.2. Privoxy's Internal Pages

Since Privoxy proxies each requested web page, it is easy for Privoxy to trap certain special URLs. In this way, we can talk directly to Privoxy, and see how it is configured, see how our rules are being applied, change these rules and other configuration options, and even turn Privoxy's filtering off, all with a web browser.

The URLs listed below are the special ones that allow direct access to Privoxy. Of course, Privoxy must be running to access these. If not, you will get a friendly error message. Internet access is not necessary either.

- Privoxy main page:

  http://config.privoxy.org/

  There is a shortcut: http://p.p/ (But it doesn't provide a fall-back to a real page, in case the request is not sent through Privoxy)
- Show information about the current configuration, including viewing and editing of actions files:

  http://config.privoxy.org/show-status
- Show the source code version numbers:

  http://config.privoxy.org/show-version
- Show the browser's request headers:

  http://config.privoxy.org/show-request
- Show which actions apply to a URL and why:

  http://config.privoxy.org/show-url-info
- Toggle Privoxy on or off. This feature can be turned off/on in the main `config` file. When toggled "off", "Privoxy" continues to run, but only as a pass-through proxy, with no actions taking place:

  http://config.privoxy.org/toggle

  Short cuts. Turn off, then on:

  http://config.privoxy.org/toggle?set=disable

  http://config.privoxy.org/toggle?set=enable

These may be bookmarked for quick reference. See next.

14.2.1. Bookmarklets

Below are some "bookmarklets" to allow you to easily access a "mini" version of some of Privoxy's special pages. They are designed for MS Internet Explorer, but should work equally well in Netscape, Mozilla, and other browsers which support JavaScript. They are designed to run directly from your bookmarks - not by clicking the links below (although that should work for testing).

To save them, right-click the link and choose "Add to Favorites" (IE) or "Add Bookmark" (Netscape). You will get a warning that the bookmark "may not be safe" - just click OK. Then you can run the Bookmarklet directly from your favorites/bookmarks. For even faster access, you can put them on the "Links" bar (IE) or the "Personal Toolbar" (Netscape), and run them with a single click.

- Privoxy - Enable
- Privoxy - Disable
- Privoxy - Toggle Privoxy (Toggles between enabled and disabled)
- Privoxy- View Status
- Privoxy - Why?

Credit: The site which gave us the general idea for these bookmarklets is www.bookmarklets.com. They have more information about bookmarklets.

14.3. Chain of Events

Let's take a quick look at how some of Privoxy's core features are triggered, and the ensuing sequence of events when a web page is requested by your browser:

- First, your web browser requests a web page. The browser knows to send the request to Privoxy, which will in turn, relay the request to the remote web server after passing the following tests:
- Privoxy traps any request for its own internal CGI pages (e.g http://p.p/) and sends the CGI page back to the browser.
- Next, Privoxy checks to see if the URL matches any "+block" patterns. If so, the URL is then blocked, and the remote web server will not be contacted. "+handle-as-image" and "+handle-as-empty-document" are then checked, and if there is no match, an HTML "BLOCKED" page is sent back to the browser. Otherwise, if it does match, an image is returned for the former, and an empty text document for the latter. The type of image would depend on the setting of "+set-image-blocker" (blank, checkerboard pattern, or an HTTP redirect to an image elsewhere).
- Untrusted URLs are blocked. If URLs are being added to the `trust` file, then that is done.
- If the URL pattern matches the "+fast-redirects" action, it is then processed. Unwanted parts of the requested URL are stripped.
- Now the rest of the client browser's request headers are processed. If any of these match any of the relevant actions (e.g. "+hide-user-agent", etc.), headers are suppressed or forged as determined by these actions and their parameters.
- Now the web server starts sending its response back (i.e. typically a web page).
- First, the server headers are read and processed to determine, among other things, the MIME type (document type) and encoding. The headers are then filtered as determined by the "+crunch-incoming-cookies", "+session-cookies-only", and "+downgrade-http-version" actions.
- If the "+kill-popups" action applies, and it is an HTML or JavaScript document, the popup-code in the response is filtered on-the-fly as it is received.
- If any "+filter" action or "+deanimate-gifs" action applies (and the document type fits the action), the rest of the page is read into memory (up to a configurable limit). Then the filter rules (from `default.filter` and any other filter files) are processed against the buffered content. Filters are applied in the order they are specified in one of the filter files. Animated GIFs, if present, are reduced to either the first or last frame, depending on the action setting.The entire page, which is now filtered, is then sent by Privoxy back to your browser.

  If neither a "+filter" action or "+deanimate-gifs" matches, then Privoxy passes the raw data through to the client browser as it becomes available.
- As the browser receives the now (possibly filtered) page content, it reads and then requests any URLs that may be embedded within the page source, e.g. ad images, stylesheets, JavaScript, other HTML documents (e.g. frames), sounds, etc. For each of these objects, the browser issues a separate request (this is easily viewable in Privoxy's logs). And each such request is in turn processed just as above. Note that a complex web page will have many, many such embedded URLs. If these secondary requests are to a different server, then quite possibly a very differing set of actions is triggered.

NOTE: This is somewhat of a simplistic overview of what happens with each URL request. For the sake of brevity and simplicity, we have focused on Privoxy's core features only.

14.4. Troubleshooting: Anatomy of an Action

The way Privoxy applies actions and filters to any given URL can be complex, and not always so easy to understand what is happening. And sometimes we need to be able to *see* just what Privoxy is doing. Especially, if something Privoxy is doing is causing us a problem inadvertently. It can be a little daunting to look at the actions and filters files themselves, since they tend to be filled with regular expressions whose consequences are not always so obvious.

One quick test to see if Privoxy is causing a problem or not, is to disable it temporarily. This should be the first troubleshooting step. See the Bookmarklets section on a quick and easy way to do this (be sure to flush caches afterward!). Looking at the logs is a good idea too. (Note that both the toggle feature and logging are enabled via `config` file settings, and may need to be turned "on".)

Another easy troubleshooting step to try is if you have done any customization of your installation, revert back to the installed defaults and see if that helps. There are times the developers get complaints about one thing or another, and the problem is more related to a customized configuration issue.

Privoxy also provides the http://config.privoxy.org/show-url-info page that can show us very specifically how actions are being applied to any given URL. This is a big help for troubleshooting.

First, enter one URL (or partial URL) at the prompt, and then Privoxy will tell us how the current configuration will handle it. This will not help with filtering effects (i.e. the "+filter" action) from one of the filter files since this is handled very differently and not so easy to trap! It also will not tell you about any other URLs that may be embedded within the URL you are testing. For instance, images such as ads are expressed as URLs within the raw page source of HTML pages. So you will only get info for the actual URL that is pasted into the prompt area -- not any sub-URLs. If you want to know about embedded URLs like ads, you will have to dig those out of the HTML source. Use your browser's "View Page Source" option for this. Or right click on the ad, and grab the URL.

Let's try an example, google.com, and look at it one section at a time in a sample configuration (your real configuration may vary):

```
 Matches for http://www.google.com:

 In file: default.action [ View ] [ Edit ]

 {+deanimate-gifs {last}
 +fast-redirects {check-decoded-url}
 +filter {refresh-tags}
 +filter {img-reorder}
 +filter {banners-by-size}
 +filter {webbugs}
 +filter {jumping-windows}
 +filter {ie-exploits}
 +hide-forwarded-for-headers
 +hide-from-header {block}
 +hide-referrer {forge}
 +session-cookies-only
 +set-image-blocker {pattern}
/

 { -session-cookies-only }
 .google.com

 { -fast-redirects }
 .google.com

In file: user.action [ View ] [ Edit ]
(no matches in this file)
```

This is telling us how we have defined our "actions", and which ones match for our test case, "google.com". Displayed is all the actions that are available to us. Remember, the + sign denotes "on". - denotes "off". So some are "on" here, but many are "off". Each example we try may provide a slightly different end result, depending on our configuration directives.

The first listing is for our `default.action` file. The large, multi-line listing, is how the actions are set to match for all URLs, i.e. our default settings. If you look at your "actions" file, this would be the section just below the "aliases" section near the top. This will apply to all URLs as signified by the single forward slash at the end of the listing -- " / ".

But we have defined additional actions that would be exceptions to these general rules, and then we list specific URLs (or patterns) that these exceptions would apply to. Last match wins. Just below this then are two explicit matches for ".google.com". The first is negating our previous cookie setting, which was for "+session-cookies-only" (i.e. not persistent). So we will allow persistent cookies for google, at least that is how it is in this example. The second turns *off* any "+fast-redirects" action, allowing this to take place unmolested. Note that there is a leading dot here -- ".google.com". This will match any hosts and sub-domains, in the google.com domain also, such as "www.google.com" or "mail.google.com". But it would not match "www.google.de"! So, apparently, we have these two actions defined as exceptions to the general rules at the top somewhere in the lower part of our `default.action` file, and "google.com" is referenced somewhere in these latter sections.

Then, for our `user.action` file, we again have no hits. So there is nothing google-specific that we might have added to our own, local configuration. If there was, those actions would over-rule any actions from previously processed files, such as `default.action`. `user.action` typically has the last word. This is the best place to put hard and fast exceptions,

And finally we pull it all together in the bottom section and summarize how Privoxy is applying all its "actions" to "google.com":

```
 Final results:

 -add-header
 -block
 -client-header-filter{hide-tor-exit-notation}
 -content-type-overwrite
 -crunch-client-header
 -crunch-if-none-match
 -crunch-incoming-cookies
 -crunch-outgoing-cookies
 -crunch-server-header
 +deanimate-gifs {last}
 -downgrade-http-version
 -fast-redirects
```

```
-filter {js-events}
-filter {content-cookies}
-filter {all-popups}
-filter {banners-by-link}
-filter {tiny-textforms}
-filter {frameset-borders}
-filter {demoronizer}
-filter {shockwave-flash}
-filter {quicktime-kioskmode}
-filter {fun}
-filter {crude-parental}
-filter {site-specifics}
-filter {js-annoyances}
-filter {html-annoyances}
+filter {refresh-tags}
-filter {unsolicited-popups}
+filter {img-reorder}
+filter {banners-by-size}
+filter {webbugs}
+filter {jumping-windows}
+filter {ie-exploits}
-filter {google}
-filter {yahoo}
-filter {msn}
-filter {blogspot}
-filter {no-ping}
-force-text-mode
-handle-as-empty-document
-handle-as-image
-hide-accept-language
-hide-content-disposition
+hide-forwarded-for-headers
+hide-from-header {block}
-hide-if-modified-since
+hide-referrer {forge}
-hide-user-agent
-inspect-jpegs
-kill-popups
-limit-connect
-overwrite-last-modified
-prevent-compression
-redirect
-send-vanilla-wafer
-send-wafer
-server-header-filter{xml-to-html}
-server-header-filter{html-to-xml}
-session-cookies-only
+set-image-blocker {pattern}
-treat-forbidden-connects-like-blocks
```

Notice the only difference here to the previous listing, is to "fast-redirects" and "session-cookies-only", which are activated specifically for this site in our configuration, and thus show in the "Final Results".

Now another example, "ad.doubleclick.net":

```
{ +block }
 ad*.

{ +block }
 .ad.

{ +block +handle-as-image }
 .[a-vx-z]*.doubleclick.net
```

We'll just show the interesting part here - the explicit matches. It is matched three different times. Two "+block" sections, and a "+block +handle-as-image", which is the expanded form of one of our aliases that had been defined as: "+block-as-image". ("Aliases" are defined in the first section of the actions file and typically used to combine more than one action.)

Any one of these would have done the trick and blocked this as an unwanted image. This is unnecessarily redundant since the last case effectively would also cover the first. No point in taking chances with these guys though ;-) Note that if you want an ad or obnoxious URL to be invisible, it should be defined as "ad.doubleclick.net" is done here -- as both a "+block" *and* an "+handle-as-image". The custom alias "`+block-as-image`" just simplifies the process and make it more readable.

One last example. Let's try "http://www.example.net/adsl/HOWTO/". This one is giving us problems. We are getting a blank page. Hmmm ...

```
Matches for http://www.example.net/adsl/HOWTO/:

In file: default.action [ View ] [ Edit ]

{-add-header
 -block
```

```
  -client-header-filter{hide-tor-exit-notation}
  -content-type-overwrite
  -crunch-client-header
  -crunch-if-none-match
  -crunch-incoming-cookies
  -crunch-outgoing-cookies
  -crunch-server-header
  +deanimate-gifs
  -downgrade-http-version
  +fast-redirects {check-decoded-url}
  -filter {js-events}
  -filter {content-cookies}
  -filter {all-popups}
  -filter {banners-by-link}
  -filter {tiny-textforms}
  -filter {frameset-borders}
  -filter {demoronizer}
  -filter {shockwave-flash}
  -filter {quicktime-kioskmode}
  -filter {fun}
  -filter {crude-parental}
  -filter {site-specifics}
  -filter {js-annoyances}
  -filter {html-annoyances}
  +filter {refresh-tags}
  -filter {unsolicited-popups}
  +filter {img-reorder}
  +filter {banners-by-size}
  +filter {webbugs}
  +filter {jumping-windows}
  +filter {ie-exploits}
  -filter {google}
  -filter {yahoo}
  -filter {msn}
  -filter {blogspot}
  -filter {no-ping}
  -force-text-mode
  -handle-as-empty-document
  -handle-as-image
  -hide-accept-language
  -hide-content-disposition
  +hide-forwarded-for-headers
  +hide-from-header{block}
  +hide-referer{forge}
  -hide-user-agent
  -inspect-jpegs
  -kill-popups
  -overwrite-last-modified
  +prevent-compression
  -redirect
  -send-vanilla-wafer
  -send-wafer
  -server-header-filter{xml-to-html}
  -server-header-filter{html-to-xml}
  +session-cookies-only
  +set-image-blocker{blank}
  -treat-forbidden-connects-like-blocks }
   /

 { +block +handle-as-image }
  /ads
```

Ooops, the "/adsl/" is matching "/ads" in our configuration! But we did not want this at all! Now we see why we get the blank page. It is actually triggering two different actions here, and the effects are aggregated so that the URL is blocked, and Privoxy is told to treat the block as if it were an image. But this is, of course, all wrong. We could now add a new action below this (or better in our own `user.action` file) that explicitly *un* blocks ("{-block}") paths with "adsl" in them (remember, last match in the configuration wins). There are various ways to handle such exceptions. Example:

```
 { -block }
  /adsl
```

Now the page displays ;-) Remember to flush your browser's caches when making these kinds of changes to your configuration to insure that you get a freshly delivered page! Or, try using `Shift+Reload`.

But now what about a situation where we get no explicit matches like we did with:

```
 { +block +handle-as-image }
 /ads
```

That actually was very helpful and pointed us quickly to where the problem was. If you don't get this kind of match, then it means one of the default rules in the first section of `default.action` is causing the problem. This would require some guesswork, and maybe a little trial and error to isolate the offending rule. One likely cause would be one of the "+filter" actions. These tend to be harder to troubleshoot. Try adding the URL for the site to one of aliases that turn off "+filter":

```
{ shop }
.quietpc.com
.worldpay.com   # for quietpc.com
.jungle.com
.scan.co.uk
.forbes.com
```

"`{ shop }`" is an "alias" that expands to "`{ -filter -session-cookies-only }`". Or you could do your own exception to negate filtering:

```
{ -filter }
# Disable ALL filter actions for sites in this section
.forbes.com
developer.ibm.com
localhost
```

This would turn off all filtering for these sites. This is best put in `user.action`, for local site exceptions. Note that when a simple domain pattern is used by itself (without the subsequent path portion), all sub-pages within that domain are included automatically in the scope of the action.

Images that are inexplicably being blocked, may well be hitting the "+filter{banners-by-size}" rule, which assumes that images of certain sizes are ad banners (works well *most of the time* since these tend to be standardized).

"`{ fragile }`" is an alias that disables most actions that are the most likely to cause trouble. This can be used as a last resort for problem sites.

```
{ fragile }
# Handle with care: easy to break
mail.google.
mybank.example.com
```

Remember to flush caches! Note that the `mail.google` reference lacks the TLD portion (e.g. ".com"). This will effectively match any TLD with `google` in it, such as `mail.google.de.`, just as an example.

If this still does not work, you will have to go through the remaining actions one by one to find which one(s) is causing the problem.