Privoxy 3.0.0 User Manual

$Id: user−manual.sgml,v 1.123.2.18 2002/08/22 23:47:58 hal9 Exp $

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

Privoxy is a web proxy with advanced filtering capabilities for protecting privacy, filtering web page content, managing cookies, controlling access, and removing ads, banners, pop−ups and other obnoxious Internet junk. Privoxy has a very 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 *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.3.0.0.

1.1. Features

In addition to Internet Junkbuster's traditional features of ad and banner blocking and cookie management, Privoxy provides new features:

- 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 content filtering (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.
- HTTP/1.1 compliant (but not all optional 1.1 features are supported).
- Support for Perl Compatible Regular Expressions in the configuration files, and generally a more sophisticated and flexible configuration syntax over previous versions.
- 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: If you have a previous Junkbuster or Privoxy installation on your system, you will need to remove it. On some platforms, this may be done for you as part of their installation procedure. (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, SuSE and Conectiva RPMs

RPMs can be installed with `rpm −Uvh privoxy−3.0.0−1.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. Note that SuSE will automatically start Privoxy in the boot process.

If you have problems with failed dependencies, try rebuilding the SRC RPM: `rpm −−rebuild privoxy−3.0.0−1.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, before installing Privoxy.

2.1.2. Debian

DEBs can be installed with `dpkg −i privoxy_3.0.0−1.deb`, 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. We do not use the registry of Windows.

2.1.4. Solaris, NetBSD, FreeBSD, HP−UX

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. 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−3.0.0` 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 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 or simply download the nightly CVS tarball.

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 (either release version or nightly CVS tarball), first unpack the source:

```
tar xzvf privoxy−3.0.0−src* [.tgz or .tar.gz]
cd privoxy−3.0.0
```

For retrieving the current CVS sources, you'll need CVS installed. Note that sources from CVS are development quality, and may not be stable, or well tested. To download CVS source:

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

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

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

```
autoheader
autoconf
./configure      # (−−help to see options)
make             # (the make from gnu, gmake for *BSD)
su
make −n install  # (to see where all the files will go)
make install     # (to really install)
```

Warning
The "make install" target is temporary quite broken! It is recommended to use a binary package, or do a source build, and manually install the components. Sorry.

If you have 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.

For more detailed instructions on how to build Redhat and SuSE 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 loose your personal changes and adjustments when updating to the latest `default.action` file we *strongly recommend* that you use `user.action` for your customization of Privoxy. See the Chapter on actions files for details.

---

3. Note to Upgraders

There are very significant changes from earlier Junkbuster versions to the current Privoxy. The number, names, syntax, and purposes of configuration files have substantially changed. Junkbuster 2.0.x configuration files will not migrate, Junkbuster 2.9.x and Privoxy configurations will need to be ported. The functionalities of the old `blockfile`, `cookiefile` and `imagelist` are now combined into the "actions files". `default.action`, is the main actions file. Local exceptions should best be put into `user.action`.

A "filter file" (typically `default.filter`) is new as of Privoxy 2.9.x, and provides some of the new sophistication (explained below). `config` is much the same as before.

If upgrading from a 2.0.x version, you will have to use the new config files, and possibly adapt any personal rules from your older files. When porting personal rules over from the old `blockfile` to the new actions files, please note that even the pattern syntax has changed. If upgrading from 2.9.x development versions, it is still recommended to use the new configuration files.

A quick list of things to be aware of before upgrading:

- The default listening port is now 8118 due to a conflict with another service (NAS).
- Some installers may remove earlier versions completely. Save any important configuration files!
- Privoxy is controllable with a web browser at the special URL: http://config.privoxy.org/ (Shortcut: http://p.p/). Many aspects of configuration can be done here, including temporarily disabling Privoxy.
- The primary configuration files for cookie management, ad and banner blocking, and many other aspects of Privoxy configuration are the actions files. It is strongly recommended to become familiar with the new actions concept below, before modifying these files. Locally defined rules should go into `user.action`.
- Some installers may not automatically start Privoxy after installation.

---

4. Quickstart to Using Privoxy

- If upgrading, from versions before 2.9.16, please back up any configuration files. See the Note to Upgraders Section.
- 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 proxy by setting the proxy configuration for address of `127.0.0.1` and port `8118`. (Junkbuster and earlier versions of Privoxy used port 8000.) See the section Starting Privoxy below for more details on this.
- Flush your browser's disk and memory caches, to remove any cached ad images.
- 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.

  See the Configuration section for more configuration options, and how to customize your installation.
- If you experience ads that slipped through, innocent images that are blocked, or otherwise feel the need to fine−tune Privoxy's behaviour, 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 "Anatomy of an Action" has hints how to debug actions that "misbehave".
- Please see the section Contacting the Developers on how to report bugs or problems with websites or to get help.
- Now enjoy surfing with enhanced 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. 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.

The actions we need to know about for ad blocking are: `block`, `handle-as-image`, and `set-image-blocker`:

- `block` − 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.
- `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.
- `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.

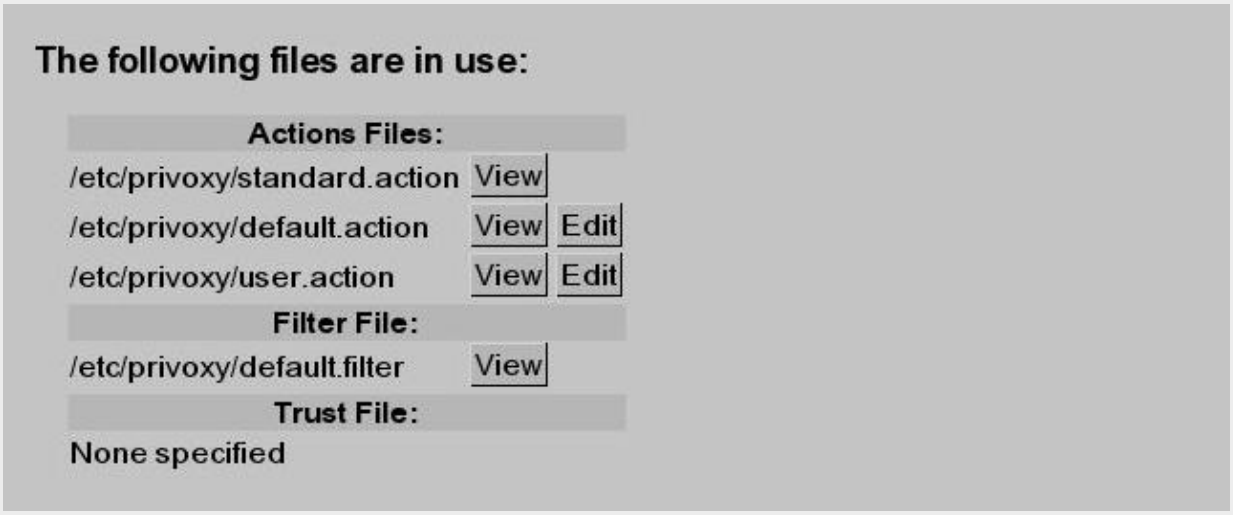
  http://<URL> − A redirect to any image anywhere of the user's choosing (advanced usage).

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. 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.

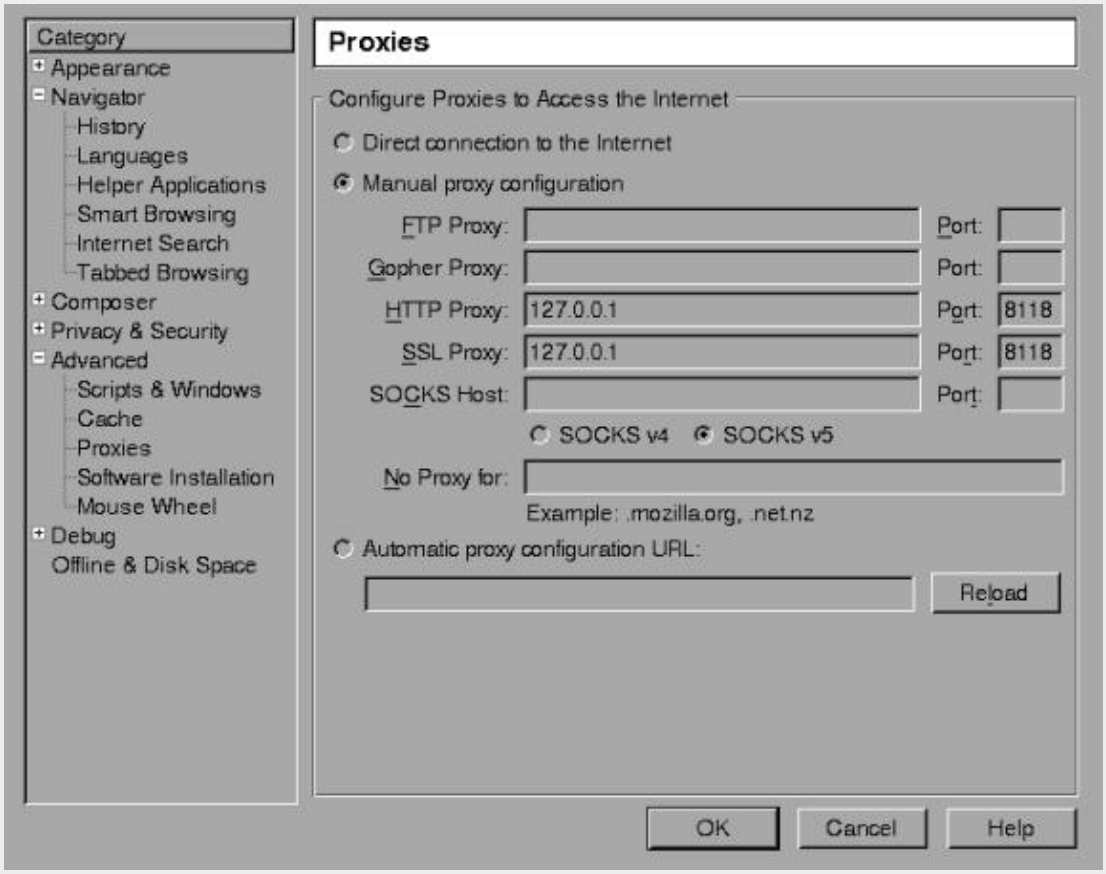
---

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 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 (Mozilla)



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

```
Edit
  |_
      Preferences
          |_
              Advanced
                  |_
                      Proxies
                          |_
                              HTTP Proxy
```

For Internet Explorer:

```
 Tools
     |_
         Internet Properties
                           |_
                                    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.

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. You are now ready to start enjoying the benefits of using Privoxy!

Privoxy 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 Conectiva

We use a script. Note that Red Hat does not start Privoxy upon booting per default. It will use the file `/etc/privoxy/config` as its main configuration file.

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

5.2. Debian

We use a script. Note that Debian 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. SuSE

We use a script. It will use the file `/etc/privoxy/config` as its main configuration file. Note that SuSE starts Privoxy upon booting your PC.

```
# rcprivoxy start
```

5.4. 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 upon booting you PC.

5.5. Solaris, NetBSD, FreeBSD, HP−UX and others

Example Unix startup command:

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

5.6. 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.

5.7. Mac OSX

During installation, Privoxy is configured to start automatically when the system restarts. To start Privoxy by hand, 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.

5.8. 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).

5.9. 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
```

5.10. 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.

- *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.

---

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.

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 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.

- `default.filter` (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.

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.

The actions files and `default.filter` 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. 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.

7.1.1. 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

  When development goes modular and multi−user, the blocker, filter, and per−user config will be stored in subdirectories of "confdir". For now, the configuration directory structure is flat, except for `confdir/templates`, where the HTML templates for CGI output reside (e.g. Privoxy's 404 error page).

7.1.2. 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

7.1.3. actionsfile

Specifies:
: The actions file(s) to use

Type of value:
: File name, relative to `confdir`, without the `.action` suffix

Default values:

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

Effect if unset:
: No actions are taken at all. Simple 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 are where all the per site and per URL configuration is done for ad blocking, cookie management, privacy considerations, etc. There is no point in using Privoxy without at least one actions file.

7.1.4. filterfile

Specifies:
: The filter file 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:
: The filter file contains content modification rules that use regular expressions. These rules permit powerful changes on the content of Web pages, e.g., you could disable your favorite JavaScript annoyances, re−write the actual displayed text, or just have some fun replacing "Microsoft" with "MicroSuck" wherever it appears on a Web page.

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

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

7.1.5. logfile

Specifies:
The log file to use
Type of value:
File name, relative to `logdir`
Default value:
logfile (Unix) *or* privoxy.log (Windows)
Effect if unset:
No log file is used, all log messages go to the console (`STDERR`).
Notes:
The windows version will additionally log to the console.

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) but in most cases you probably will never look at it.

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, a **logrotate** script has been included.

On SuSE Linux systems, you can place a line like "/var/log/privoxy.* +1024k 644 nobody.nogroup" in `/etc/logfiles`, with the effect that cron.daily will automatically archive, gzip, and empty the log, when it exceeds 1M size.

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

7.1.6. jarfile

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

7.1.7. trustfile

Specifies:
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 whole trust mechanism is turned off.
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 named in the trustfile. You can also mark sites as trusted referrers (with +), with the effect that access to untrusted sites will be granted, if a link from a trusted referrer was used. The link target will then be added to the "trustfile". Possible applications include limiting Internet access for children.

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

7.2. 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.

7.2.1. 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 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. For multi−user setups, you could provide a copy on a local webserver for all your users and use the corresponding URL here.

Examples:

Unix, in local filesystem:

```
user−manual  file:///usr/share/doc/privoxy−3.0.0/user−manual/
```

Any platform, on local webserver (called "local−webserver"):

```
user-manual  http://local-webserver/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.

7.2.2. 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 URL 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* above.)

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!

7.2.3. admin-address

Specifies:
An email address to reach the proxy 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.

7.2.4. 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 ;-)

7.3. 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 to the *logfile*.

Type of value:

Integer values

Default value:

12289 (i.e.: URLs plus informational and warning messages)

Effect if unset:

Nothing gets logged.

Notes:

The available debug levels are:

```
debug         1 # show each GET/POST/CONNECT request
debug         2 # show each connection status
debug         4 # show I/O status
debug         8 # show header parsing
debug        16 # log all data into the logfile
debug        32 # debug force feature
debug        64 # debug regular expression filter
debug       128 # debug fast 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 highly 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).

The reporting of *fatal* errors (i.e. ones which crash Privoxy) is always on and cannot be disabled.

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

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 debug purposes and you should never need to use it. *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 turn off the `enable−edit−actions` and `enable−remote−toggle` options!
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. behave like a normal, content−neutral proxy where all ad blocking, filtering, etc are disabled. See `enable-remote-toggle` below. This is not really useful anymore, since toggling is much easier via the web interface than via editing the `conf` file.

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:
1
Effect if unset:
The web−based toggle feature is disabled.
Notes:
When toggled off, Privoxy acts like a normal, content−neutral proxy, i.e. it acts as if none of the actions applied to any URL.

For the time being, 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 you must have compiled Privoxy with support for this feature, otherwise this option has no effect.

7.4.4. enable−edit−actions

Specifies:
Whether or not the web−based actions file editor may be used
Type of value:
0 or 1
Default value:
1
Effect if unset:
The web−based actions file editor is disabled.

Notes:

For the time being, 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. So this option is *not recommended* for multi-user environments with untrusted users.

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

7.4.5. 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 this proxy 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, then the Privoxy talks only 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.

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:

```
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 www.dirty-stuff.example.com:

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

7.4.6. 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. It can be used to better protect privacy and confidentiality when accessing specific domains by routing requests to those domains through an anonymous public proxy (see e.g. http://www.multiproxy.org/anon_list.htm) Or to use a caching proxy to speed up browsing. Or chaining to a parent proxy may be necessary because the machine that Privoxy runs on has no direct Internet access.

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 anonymizing proxy, except SSL on port 443 (which it doesn't handle):

```
forward   /      anon-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.example-isp.net:8000
forward   .example-isp.net   .
```

7.5.2. forward-socks4 and forward-socks4a

Specifies:

Through which SOCKS proxy (and 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.example-isp.net:80
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  .
```

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

host-a:

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

host-b:

```
forward    /           .
forward    .isp-a.net  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 chain 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 for 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
```

7.6. 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 determine 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 three such files included with Privoxy (as of version 2.9.15), 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 for users everywhere. This is the file that the developers are keeping updated, and making available to users.
- `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 by the web based editor, to set various pre−defined sets of rules for the default actions section in `default.action`. These have increasing levels of aggressiveness *and have no influence on your browsing unless you select them explicitly in the editor*. It is not recommend to edit this file.

The list of actions files to be used are defined in the main configuration file, and are processed in the order they are defined. The content of these can all be viewed and edited from http://config.privoxy.org/show−status.

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 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 that 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, 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. 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 kill popup windows per default, you'll have to make exceptions from that rule for sites that you regularly use and that require popups for actually useful content, 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. 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".

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

8.3. How Actions are Applied to URLs

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 patterns, each on a separate line.

To determine which actions apply to a request, the URL of the request is compared to all patterns in each action file file. Every time it matches, the list of applicable actions for the URL is incrementally updated, using the heading of the section in which the pattern is located. If multiple matches for the same URL 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.

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

More detail on this is provided in the Appendix, Anatomy of an Action.

8.4. Patterns

Generally, a pattern has the form `<domain>/<path>`, where both the `<domain>` and `<path>` are optional. (This is why the pattern `/` matches all URLs).

`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.

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

`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.

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

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 that *ENDS* in `.example.com`

`www.`
: matches any domain that *STARTS* with `www.`

`.example.`
: matches any domain that *CONTAINS* `.example.` (Correctly speaking: It matches any FQDN that contains `example` as a domain.)

Additionally, there are wild−cards that you can use in the domain names themselves. They work pretty similar to shell wild−cards: "*" stands for zero or more arbitrary characters, "?" stands for any single character, you can define character classes in square brackets and all of that 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`.

8.4.2. The Path Pattern

Privoxy uses Perl compatible regular expressions (through the PCRE library) for matching the path.

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://www.perldoc.com/perl5.6/pod/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.

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.

There are three classes of actions:

- 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 1.0 }`
- 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-anonymizing 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 actions always over-ride earlier ones. 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). 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 pattern to match more than one pattern and thus more than one set of actions!

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 obnoxious content
Effect:
Requests for URLs to which this action applies are blocked, i.e. the requests are not forwarded to the remote server, but answered locally with a substitute page or image, as determined by the `handle-as-image` and `set-image-blocker` 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.

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
```

8.5.3. crunch-incoming-cookies

Typical use:

Prevent the web server from setting any 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* cookies. For *outgoing* cookies, use `crunch-outgoing-cookies`. Use *both* to disable 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.

Example usage:

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

8.5.4. crunch-outgoing-cookies

Typical use:

Prevent the web server from reading any 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* cookies. For *incoming* cookies, use `crunch-incoming-cookies`. Use *both* to disable 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
```

8.5.5. 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}
```

8.5.6. 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 (optional) HTTP/1.1 features are supported yet, so there is a chance you might need this action.
Example usage (section):

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

8.5.7. fast−redirects

Typical use:
Fool some click−tracking scripts and speed up indirect links
Effect:
Cut off all but the last valid URL from requests.
Type:
Boolean.
Parameter:

N/A

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://some.place/click−tracker.cgi?target=http://some.where.else*.

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 ask 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. It is likely to break some sites. You should expect to need possibly many exceptions to this action, if it is enabled by default in `default.action`. Some sites just don't work without it.

Example usage:

```
{+fast−redirects}
```

8.5.8. filter

Typical use:

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

Effect:

Text documents, including HTML and JavaScript, to which this action applies, are filtered on−the−fly through the specified regular expression based substitutions.

Type:

Parameterized.

Parameter:

The name of a filter, as defined in the filter file (typically `default.filter`, set by the `filterfile` option in the config file). Filtering can be completely disabled without the use of parameters.

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.

This is potentially a very powerful feature! But "rolling your own" filters requires a knowledge of regular expressions and HTML.

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.

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 are not filtered.

At this time, Privoxy cannot (yet!) uncompress compressed documents. If you want filtering to work on all documents, even those that would normally be sent compressed, use the

prevent-compression action in conjunction with `filter`.

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!

Example usage (with filters from the distribution `default.filter` file):

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

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

+filter{banners-by-size}     # Kill banners based on their size for this page (very effici

+filter{banners-by-link}     # Kill banners based on the link they are contained in (exper

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

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

+filter{popups}              # Kill all popups in JS and HTML

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

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

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

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

+filter{nimda}               # Remove Nimda (virus) code.

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

+filter{crude-parental}      # Kill all web pages that contain the words "sex" or "warez"
```

```
+filter{js-events}           # Kill all JS event bindings (Radically destructive! Only for
```

8.5.9. handle-as-image

Typical use:

Mark URLs as belonging to images (so they'll be replaced by images *if they get blocked*)

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
```

8.5.10. hide-forwarded-for-headers

Typical use:

Improve privacy by hiding the true source of the request

Effect:

Deletes any existing "X-Forwarded-for:" HTTP header from client requests, and prevents adding a new one.

Type:

Boolean.

Parameter:

N/A

Notes:

It is fairly safe to leave this on.

This action is scheduled for improvement: It should be able to generate forged "X−Forwarded−for:" headers using random IP addresses from a specified network, to make successive requests from the same client look like requests from a pool of different users sharing the same proxy.

Example usage:

```
+hide−forwarded−for−headers
```

8.5.11. 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}
```

8.5.12. 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:

- ◊ "block" to delete the header completely.
- ◊ "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:

"forge" is the preferred option here, since some servers will not send images back otherwise, in an

attempt to prevent their valuable content from being embedded elsewhere (and hence, without being surrounded by *their* banners).

`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/}
```

8.5.13. hide-user-agent

Typical use:

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 breaks many web sites that depend on looking at this header in order to customize their content for different browsers (which, by the way, is *NOT* a smart way to do that!).

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 :-).

This action is scheduled for improvement.

Example usage:

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

8.5.14. kill-popups

Typical use:

Eliminate those annoying pop-up windows

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 easily confused with the built−in, hardwired `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{popups}` does.

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 is a dangerous business. Many shops and banks rely on pop−ups to display forms, shopping carts etc, and killing only the unwanted pop−ups would require artificial intelligence in Privoxy. 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.
Example usage:

```
+kill−popups
```

8.5.15. limit−connect

Typical use:
Prevent abuse of Privoxy as a TCP proxy relay
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 can be a big security hole, since CONNECT−enabled proxies can be abused as TCP relays very easily.

If you don't know what any of this means, there probably is no reason to change this one, since the default is already very restrictive.
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
+limit-connect{-}                     # All ports are OK (gaping security hole!)
```

8.5.16. prevent-compression

Typical use:

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

Effect:

Adds a header to the request that asks for uncompressed 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 for the `filter`, `deanimate-gifs` and `kill-popups` actions to work, Privoxy needs access to the uncompressed data. Unfortunately, Privoxy can't yet(!) uncompress, filter, and re-compress the content on the fly. So if you want to ensure that all websites, including those that normally compress, can be filtered, you need to use this action.

This will slow down transfers from those websites, though. If you use any of the above-mentioned actions, you will typically want to use `prevent-compression` in conjunction with them.

Note that some (rare) ill-configured sites don't handle requests for uncompressed documents correctly (they send an empty document body). If you use `prevent-compression` per default, you'll have to add exceptions for those sites. See the example for how to do that.

Example usage (sections):

```
# Set default:
#
{+prevent-compression}
/ # Match all sites

# Make exceptions for ill sites:
#
{-prevent-compression}
www.debianhelp.org
www.pclinuxonline.com
```

8.5.17. 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
```

8.5.18. 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
```

8.5.19. 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.

Example usage:

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

8.5.20. 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).

  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 devil:

```
+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}
```

8.5.21. 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. This is likely to change in future versions of Privoxy.

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
mercy−for−cookies   = −crunch−all−cookies −session−cookies−only

# These aliases define combinations of actions
# that are useful for certain types of sites:
#
fragile     = −block −crunch−all−cookies −filter −fast−redirects −hide−referer −kill−popups
shop        = −crunch−all−cookies −filter{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
.nytimes.com

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

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

Aliases like "shop" and "fragile" are often used for "problem" sites that require some actions 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 <developers@privoxy.org>
```

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

# These aliases define combinations of actions
# that are useful for certain types of sites:
#
fragile     = -block -crunch-all-cookies -filter -fast-redirects -hide-referer -kill-popups
shop        = mercy-for-cookies -filter{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 real need to disable any actions here, but we will do that nonetheless, to have a complete listing for your reference. (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:
############################################################################
 { \
 -add-header \
 -block \
 -crunch-incoming-cookies \
 -crunch-outgoing-cookies \
 +deanimate-gifs \
 -downgrade-http-version \
 +fast-redirects \
 +filter{html-annoyances} \
 +filter{js-annoyances} \
 -filter{content-cookies} \
 +filter{popups} \
 +filter{webbugs} \
 -filter{refresh-tags} \
 -filter{fun} \
 +filter{nimda} \
 +filter{banners-by-size} \
 -filter{banners-by-link} \
 -filter{img-reorder} \
 -filter{shockwave-flash} \
```

```
-filter{crude-parental} \
-filter{js-events} \
-handle-as-image \
+hide-forwarded-for-headers \
+hide-from-header{block} \
+hide-referrer{forge} \
-hide-user-agent \
-kill-popups \
-limit-connect \
+prevent-compression \
-send-vanilla-wafer \
-send-wafer \
+session-cookies-only \
+set-image-blocker{pattern} \
}
/ # forward slash will match *all* potential URL patterns.
```

The default behavior is now set. Note that some actions, like not hiding the user agent, are part of a "general policy" that applies universally and won't get any exceptions defined later. Other choices, like not blocking (which is *understandably* the default!) need exceptions, i.e. we need to specify explicitly what we want to block in later sections. We will also want to make exceptions from our general pop-up-killing, and use our defined aliases for that.

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
```

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
```

Then, there are sites which rely on pop-up windows (yuck!) to work. Since we made pop-up-killing our default above, we need to make exceptions now. Mozilla users, who can turn on smart handling of unwanted pop-ups in their browsers, can safely choose -`filter{popups}` (and -`kill-popups`) above and hence don't need this section. Anyway, disabling an already disabled action doesn't hurt, so we'll define our exceptions regardless of what was chosen in the defaults section:

```
# These sites require pop-ups too :(
```

```
#
{ -kill-popups -filter{popups} }
.dabs.com
.overclockers.co.uk
.deutsche-bank-24.de
```

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
bs*.einets.com
.qkimg.net
```

One of the most important jobs of Privoxy is to block banners. A huge bunch of them are already "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 a bunch of 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
```

You wouldn't believe 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)
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
.sourceforge.net
```

The actual `default.action` is of course 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@foobar.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:

```
# (Re-)define aliases for this file:
#
{{alias}}
-crunch-all-cookies = -crunch-incoming-cookies -crunch-outgoing-cookies
mercy-for-cookies   = -crunch-all-cookies -session-cookies-only
fragile     = -block -crunch-all-cookies -filter -fast-redirects -hide-referer -kill-popups
shop        = mercy-for-cookies -filter{popups} -kill-popups
allow-ads   = -block -filter{banners-by-size} # (see below)
```

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 `mercy-for-cookies` alias defined above does exactly that, i.e. it disables crunching of cookies in any direction, and processing of cookies to make them temporary.

```
{ mercy-for-cookies }
sunsolve.sun.com
slashdot.org
.yahoo.com
.msdn.microsoft.com
.redhat.com
```

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

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

While browsing the web with Privoxy you noticed some ads that sneaked through, but you were too lazy to report them through our fine and easy feedback system, so you have added them here:

```
{ +block }
www.a-popular-site.com/some/unobvious/path
another.popular.site.net/more/junk/here/
```

Note that, assuming the banners in the above example have regular image extensions (most do), +handle-as-image need not be specified, since all URLs ending in these extensions will already have been tagged as images in the relevant section of `default.action` by now.

Then 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:

```
{ fragile }
.forbes.com
```

You like the "fun" text replacements in `default.filter`, but it is disabled in the distributed actions file. (My colleagues on the team just don't have a sense of humour, that's why! ;-). 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.

Finally, you might think 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} above.

---

9. The Filter File

All text substitutions that can be invoked through the `filter` action must first be defined in the filter file, which is typically called `default.filter` and which can be selected through the `filterfile` config option.

Typical reasons for doing such substitutions 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. The possibilities are endless.

Filtering works on any text−based document type, including plain text, HTML, JavaScript, CSS etc. (all `text/*` MIME types). Substitutions are made at the source level, so if you want to "roll your own" filters, you should be familiar with HTML syntax.

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 the *keyword* `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.

A 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 man page 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" 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 backreference to the first parenthesis just like `$1` above, with the difference that in the *pattern*, a backslash indicates a backreference, 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?

---

10. Templates

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. You can edit the templates with a normal text editor, should you want to customize them. (*Not recommended for the casual user*). 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 in 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.

11.2. Report Bugs

Please report all bugs *only* 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.

Please try to verify that it is a Privoxy bug, and not a browser or site bug first. If unsure, try toggling off Privoxy, and see if the problem persists. The appendix of the user manual also has helpful information on action debugging. If you are using your own custom configuration, please try the stock configs to see if the problem is configuration related.

If not using the latest version, chances are that the bug has 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 your bug, but this is not required for reporting.

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. Report Ads or Other Actions−Related Problems

Please send feedback on ads that slipped through, innocent images that were blocked, and any other problems relating to the `default.action` file through our actions feedback mechanism located at http://www.privoxy.org/actions/. On this page, you will also find a bookmark which will take you back there from any troubled site and even pre−fill the form!

New, improved `default.action` files will 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.

11.5. 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

In the beginning, there was the Internet Junkbuster, by Anonymous Coders and Junkbusters Corporation. It 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 users to consume ads, give up autonomy over their browsing, and for spying on them, kept 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 release, 3.0, was released August, 2002.

12.3. Authors

Current Project Developers:

```
Jon Foster
Andreas Oesterhelt
Stefan Waldherr

Thomas Steudten
Rodney Stromlund
```

Current Project Contributors:

```
Rodrigo Barbosa (RPM specfiles)
Moritz Barsnick
Hal Burgiss (docs)
Karsten Hopp (Red Hat)
Alexander Lazic
Gábor Lipták
Guy
Haroon Rafique
Roland Rosenfeld (Debian)
Georg Sauthoff (Gentoo)
David Schmidt (OS/2, Mac OSX ports)
Joerg Strohmayer (Amiga)
Sarantis Paskalis
```

Based in part on code originally developed by:

```
Junkbusters Corp.
Anonymous Coders
```

Thanks to the many people who have tested Privoxy, reported bugs, or made suggestions. These include (in alphabetical order):

```
Ken Arromdee
Devin Bayer
Reiner Buehl
Andrew J. Caines
Clifford Caoile
Michael T. Davis
Peter E
Aaron Hamid
Magnus Holmgren
Daniel Leite
Paul Lieverse
Roberto Ragusa
Maynard Riley
Bart Schelstraete
Darren Wiebe
```

jwz

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://www.privoxy.org/actions/, to submit "misses" to the developers.

http://cvs.sourceforge.net/cgi−bin/viewcvs.cgi/ijbswa/contrib/, cool and fun ideas from Privoxy users.

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://www.waldherr.org/junkbuster/, Stefan Waldherr's version of Junkbuster, from which Privoxy was derived.

http://privacy.net/analyze/, a useful site to check what information about you is leaked while you browse the web.

http://www.squid−cache.org/, a very popular caching proxy, which is often used together with Privoxy.

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.

A 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://www.perldoc.com/perl5.6/pod/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. In this case, "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 − Submit Actions File Feedback
- 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 the basic sequence of events when a web page is requested by your browser and Privoxy is on duty:

- 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" is then checked and if it does not match, an HTML "BLOCKED" page is sent back. Otherwise, if it does match, an image is returned. The type of image depends 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 and related data).
- 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 a "+filter" 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`) are processed against the buffered content. Filters are applied in the order they are specified in the `default.filter` file. 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 "+filter" 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 (probably 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 new request. And each such request is in turn processed as above. Note that a complex web page may have many such embedded URLs.

14.4. 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.

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 the `default.filter` file 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:

```
 Matches for http://google.com:

 In file: default.action [ View ] [ Edit ]

{−add−header
 −block
 −crunch−outgoing−cookies
 −crunch−incoming−cookies
 +deanimate−gifs{last}
 −downgrade−http−version
 +fast−redirects
 −filter{popups}
 −filter{fun}
```

```
 -filter{shockwave-flash}
 -filter{crude-parental}
 +filter{html-annoyances}
 +filter{js-annoyances}
 +filter{content-cookies}
 +filter{webbugs}
 +filter{refresh-tags}
 +filter{nimda}
 +filter{banners-by-size}
 +hide-forwarded-for-headers
 +hide-from-header{block}
 +hide-referer{forge}
 -hide-user-agent
 -handle-as-image
 -kill-popups
 -limit-connect
 +prevent-compression
 -send-vanilla-wafer
 -send-wafer
 +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 tells us how we have defined our "actions", and which ones match for our example, "google.com". The first listing is any matches for the `standard.action` file. No hits at all here on "standard". Then next is "default", or 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 can define additional actions that would be exceptions to these general rules, and then 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. 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". So, apparently, we have these two actions defined 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.

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
```

```
-crunch-outgoing-cookies
-crunch-incoming-cookies
+deanimate-gifs{last}
-downgrade-http-version
-fast-redirects
-filter{popups}
-filter{fun}
-filter{shockwave-flash}
-filter{crude-parental}
+filter{html-annoyances}
+filter{js-annoyances}
+filter{content-cookies}
+filter{webbugs}
+filter{refresh-tags}
+filter{nimda}
+filter{banners-by-size}
+hide-forwarded-for-headers
+hide-from-header{block}
+hide-referer{forge}
-hide-user-agent
-handle-as-image
-kill-popups
-limit-connect
+prevent-compression
-send-vanilla-wafer
-send-wafer
-session-cookies-only
+set-image-blocker{pattern}
```

Notice the only difference here to the previous listing, is to "fast−redirects" and "session−cookies−only".

Now another example, "ad.doubleclick.net":

```
{ +block +handle-as-image }
 .ad.doubleclick.net

{ +block +handle-as-image }
 ad*.

{ +block +handle-as-image }
 .doubleclick.net
```

We'll just show the interesting part here, the explicit matches. It is matched three different times. Each as an "+block +handle−as−image", which is the expanded form of one of our aliases that had been defined as: "+imageblock". ("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 "+imageblock" just simplifies the process and make it more readable.

One last example. Let's try "http://www.rhapsodyk.net/adsl/HOWTO/". This one is giving us problems. We are getting a blank page. Hmmm ...

```
Matches for http://www.rhapsodyk.net/adsl/HOWTO/:
```

```
In file: default.action [ View ] [ Edit ]

{−add−header
 −block
 −crunch−incoming−cookies
 −crunch−outgoing−cookies
 +deanimate−gifs
 −downgrade−http−version
 +fast−redirects
 +filter{html−annoyances}
 +filter{js−annoyances}
 +filter{kill−popups}
 +filter{webbugs}
 +filter{nimda}
 +filter{banners−by−size}
 +filter{hal}
 +filter{fun}
 +hide−forwarded−for−headers
 +hide−from−header{block}
 +hide−referer{forge}
 −hide−user−agent
 −handle−as−image
 +kill−popups
 +prevent−compression
 −send−vanilla−wafer
 −send−wafer
 +session−cookies−only
 +set−image−blocker{blank} }
  /

{ +block +handle−as−image }
 /ads
```

Ooops, the "/adsl/" is matching "/ads"! But we did not want this at all! Now we see why we get the blank page. We could now add a new action below this that explicitly does *not* block ("{−block}") paths with "adsl". There are various ways to handle such exceptions. Example:

```
{ −block }
 /adsl
```

Now the page displays ;−) Be sure to flush your browser's caches when making such changes. 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 telling 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 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}
.forbes.com
```

This would turn off all filtering for that site. This would probably be most appropriately put in `user.action`, for local site exceptions.

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. This can be used as a last resort for problem sites. Remember to flush caches! 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.