Erebot

Release 0.6.0

September 18, 2016

Contents

Erebot is a modular IRC bot written in PHP. It is known to work under both Linux and Windows (contact us on GitHub if you managed to use it on MacOSX) and works on a wide range of PHP versions from PHP 5.3.3 onwards.

You can find more information about this project on the following websites:

- https://www.erebot.net/—The official website.
- https://github.com/Erebot/Erebot/—Project page on GitHub.
- https://packages.erebot.net/—Repository for the project.
- https://ci.erebot.net/—Our Continuous Integration server.
- http://travis-ci.org/Erebot/Erebot—Travis Continuous Integration.

Contents:

CHAPTER 1

Prerequisites

This page assumes that the reader has a working PHP setup (either installed using some distribution's package manager or manually) and lists the dependencies required to use Erebot. In case you compiled PHP yourself, you may need to recompile it to include additional extensions (see the list of required *PHP dependencies* for more information).

As of this writing, Erebot is known to work on all PHP versions since PHP 5.2.1. Also, Erebot should run correctly on both Windows (XP or later) and most Linux distributions. The code is tested using an automated process on several operating systems, as reflected by our Continuous Integration server.

Table of Contents

1.1 How to read this page

Each dependency is associated with a "profile" which indicates who might be interested in installing that particular dependency. Currently, the following profiles are used:

developer Someone who contributes to the project, eg. by sending patches or pull requests. The developer profile is usually a superset of the end-user profile as developers tend to run the bot to test their work.

end-user Someone who wants to run the bot, ie. have it connect to and interact with some IRC server. Only a few dependencies are required for this profile.

packager Someone who creates (PHAR) packages from Erebot's source code. This profile does not include distribution maintainers.

You only need to install the dependencies associated with the profile that best matches what you plan on doing.

1.2 System dependencies

The following table lists system dependencies. Make sure to read the installation instructions relevant to your system **before** you start the installation:

- *Instructions for Linux users*
- *Instructions for Windows users*

For each profile, a "yes" on a dependency's row indicates that users of the given profile **MUST** install that dependency (requirement). Optional dependencies are indicated using footnotes which state when the dependency may be of any interest for the given profile.

For apt-based systems like Debian/Ubuntu, an installation link is also provided.

Table 1.1: System dependencies for Erebot

Dependency	APT link	*Developer*	*Packager*	*End-user*	Description
doxygen	Install using apt	yes	yes		`doxygen` is needed if you plan to generate the documentation from Erebot's source files. We recommend version 1.7.2 or later as Erebot makes heavy use of PHP type-hinting and `doxygen` did not support that until 1.7.2.
gettext	Install using apt	yes	yes		The `gettext` package provides the `xgettext` command-line program used to extract messages marked for translation. **Note:** This is **NOT** the same as the PHP `gettext` extension.
xmlstarlet	Install using apt		yes		`xmlstarlet` is a CLI (Command-Line Interface) tool that simplifies XML files editing. We use it during packaging to set various settings in the `package.xml` file.

1.2.1 Instructions for Linux users

It is assumed that the reader has a working package manager which can be used to install those dependencies, usually by issuing one of the following commands **as a privileged user**, followed by the name of the package that provides the dependency:

```
root@localhost:~# # For apt-based distributions (eg. Debian, Ubuntu).
root@localhost:~# apt-get install <package>
```

```
root@localhost:~# # For yum-based distributions (eg. Fedora, RedHat, CentOS).
root@localhost:~# yum install <package>

root@localhost:~# # For urpmi-based distributions (eg. Mandriva).
root@localhost:~# urpmi <package>

root@localhost:~# # For Zypper-based distributions (eg. SuSE)
root@localhost:~# zypper install <package>
```

1.2.2 Instructions for Windows users

As Windows lacks a central package manager, each dependency must be downloaded and installed separately, using its specific procedure.

For Doxygen, this is easy, just go to Doxygen's download page. The same goes for XMLStarlet whose Windows version can easily be downloaded from their download page.

Installing gettext for Windows is a little bit thougher. First, go to http://ftp.gnome.org/pub/gnome/binaries/win32/dependencies/ and download the latest version (0.18.1.1-2 as of this writing) of the following archives :

- gettext-runtime-dev
- gettext-runtime
- gettext-tools-dev

Unzip each of these files to the same target folder (eg. `C:\gettext`).

Note: So as to avoid potential issues, we recommend that you unzip the files in a folder whose name is both short (eg. your disk drive's root) and does not contain any special character (including spaces).

Once you are done, point your system's `PATH` environment variable to that folder's `bin` subdirectory (ie. `C:\gettext\bin`). The remaining folders (lib, include, share, etc.) are not required and can safely be removed if disk space is an issue.

This setup is what we use to test Erebot on Windows on our Continuous Integration server.

1.3 PHP dependencies

There are two kinds of dependencies:

PEAR packages These packages contain (reusable) PHP code. They are downloaded from the PHP Extension and Application Repository.

PECL packages These packages contain code (usually written in C) that extends PHP with new features or changes the behaviour of existing features. They are downloaded from the PHP Extension Community Library.

For each dependency, a short description as well as the profiles that are likely to be interested in installing that dependency are listed.

1.3.1 Compatible installers

To install Erebot's PHP dependencies, you will need a compatible installer. There are currently two of them:

pear The original installer, meant to install both PEAR and PECL packages. The simplest way to install pear is to grab a copy of go-pear.phar and run this command from a shell:

```
$ php go-pear.phar
```

Then, to install a dependency using pear, run the following command:

```
$ pear install <dependency>
```

Pyrus (highly experimental, see below) Successor for pear, meant to replace it someday. Pyrus provides the means to install and manage installations for packages built using package.xml version 2.0 or newer. Pyrus is redesigned from the ground up for PHP 5.3 or newer, and provides significant improvements over the older PEAR Installer. The latest version can be downloaded from this link.

To install a dependency using Pyrus, run the following command:

```
$ php pyrus.phar install <dependency>
```

Warning: At the time of this writing, Pyrus is still in development, with only alpha releases currently available. Pyrus may corrupt your system when using its default configuration and is also known to install pear packages incorrectly under certain circumstances. Unless you know exactly what you are doing, we recommend that you stick to the regular pear tool to install Erebot. See https://github.com/pyrus/Pyrus/issues/8 and https://github.com/pyrus/Pyrus/issues/26 for more information.

1.3.2 PECL extensions

The following table lists the PECL extensions needed to use Erebot. You may notice that most of these extensions are actually part of PHP Core.

For each profile, a "yes" on a dependency's row indicates that users of the given profile **MUST** install that dependency (requirement). Optional dependencies are indicated using footnotes which state when the dependency may be of any interest for the given profile.

Unless you have a good reason not to (such as when testing backward compatibility), we recommend that you always install the latest version available for each dependency.

Table 1.2: PECL extensions used by Erebot

Dependency	*Developer*	*Packager*	*End-user*	Description
DOM	yes		yes	The DOM extension parses an XML (eXtensible Markup Language) document into a DOM (Document Object Model), making it easier to work with from a developer's point of view.
intl	yes	yes	yes	Provides several helper classes to ease work on I18N (internationalization) in PHP applications.
libxml	yes		yes	This extension is a thin wrapper above the C libxml2 library and is used by other extensions (DOM, SimpleXML, XML, etc.) that deal with XML documents.
openssl			[1]	Provides SSL/TLS support (secure communications) for PHP.
pcntl			[2]	Process management using PHP. The functions provided by this extension can be used to communicate with other processes from PHP (using signals) and to exercise some sort of control over them.
Phar		[3]	[4]	This extension is used to create or access a PHP Archive (phar).
POSIX			[5]	Provides access to several functions only featured by POSIX-compliant operating systems.
Reflection	yes		yes	This extension makes it possible for some PHP code to inspect its own structure.
SimpleXML	yes		yes	Wrapper around libxml2 designed to make working with XML documents easier.
sockets	yes		yes	This extensions provides networking means for PHP applications.
SPL	yes		yes	The Standard PHP Library provides several functions and classes meant to deal with common usage patterns, improving code reuse.
xdebug	[6]			Debugging execution of PHP code is made possible by this extension. It can also be used to retrieve some metrics on the code (like code coverage information).
XSL	yes			The XSL extension implements the XSL standard, performing XSLT transformations using the libxslt library.
mbstring or iconv or recode or XML	yes		yes	These extensions make it possible to re-encode some text (also known as transcoding) from one encoding to another. `mbstring` and `iconv` support a wider set of encodings than the other extensions and are thus recommended.

1.3.3 PEAR packages

The following table lists the PEAR packages needed to use Erebot.

For each profile, a "yes" on a dependency's row indicates that users of the given profile **MUST** install that dependency (requirement). Optional dependencies are indicated using footnotes which state when the dependency may be of any interest for the given profile.

Unless you have a good reason not to (such as when testing backward compatibility), we recommend that you always install the latest version available for each dependency.

[1] Needed if you want to connect to IRC servers using a secure (encrypted) connection. Required when running Erebot from a PHAR archive (used to check the archive's origin and integrity).

[2] Required for daemonization and to change user/group information upon startup. Not available on Windows.

[3] Only required to package Erebot as a `.phar` archive.

[4] Only required to run Erebot from a `.phar` archive.

[5] Required to change user/group information upon startup. Not available on Windows.

[6] Only required to run the test suite.

Table 1.3: PEAR packages used by Erebot

Dependency	Developer	Packager	End-user	Description
pear.pdepend.org/PHP_Depend	[7]			PHP Depend gives several metrics on PHP code such as adherence between classes.
pear.phing.info/Phing >= 2.4.3	yes	yes		PHING (PHing Is Not GNU make) is a PHP project build tool based on Apache Ant. It is heavily used by Erebot which provides phing targets for most operations you may use.
pear.php.net/Console_CommandLine	yes	yes	yes	Parses command line arguments. This is used by Erebot to provide options for the bot (eg. to change the path to the configuration file, to start the bot in the background, etc.).
pear.php.net/File_Gettext	yes		yes	Erebot uses this PEAR package to handle I18N. It can be used to parse gettext translation catalogs, like the ones provided with Erebot.
pear.php.net/PHP_CodeSniffer	yes[8]			This package tokenizes PHP files and detects violations of a defined set of coding standards. It is used by Erebot developers to make sure new patches comply with Erebot's coding standard.
pear.php.net/PHP_ParserGenerator	yes	yes		This package is is a port of the Lemon parser generator for PHP and is used by Erebot and its modules to create parsers for several grammars (eg. to parse expressions in styles).
pear.phpmd.org/PHP_PMD	[9]			The PHP Mess Detector parses PHP files to detect overly complex code patterns, making it easier for developpers to refactor their code and to improve its readability.
pear.phpunit.de/phpcpd	[10]			The PHP Copy/Paste Detector detects abusive duplication of PHP code.
pear.phpunit.de/PHPUnit >= 3.4.0	[11]			PHP unit test framework used by Erebot. Pull requests should generally contain one or more unit test before they can be considered for review.

[7] Required to use the `qa_depend` phing target.
[8] Required to use the `qa_codesniffer` phing target, which should **ALWAYS** be called before submitting a patch.
[9] Required to use the `qa_mess` phing target.
[10] Required to use the `qa_duplicates` phing target.
[11] Required to use any of the `qa_coverage`, `qa_test`, `test` or `tests` phing targets.

CHAPTER 2

Installation

This pages contains instructions on how to install Erebot on your machine. There are several ways to achieve that. Each method is described below.

Warning: You cannot mix the different methods. Especially, **you must use the same method to install modules as the one you selected for Erebot itself**.

Note: We recommend using the *PHAR installation* method or the **'composer installation'_** method, depending on whether your project already uses **'Composer'_** or not. Installation from sources is reserved for advanced installations (mainly for Erebot's developers).

2.1 Installation using PHAR archives

A PHAR archive is simply a way of bundling all the necessary files in one big file. However, PHAR's archive does not contain any module. Thus, to get a working installation, you must install additional Erebot modules. At a minimum, this includes: Erebot_Module_IrcConnector, Erebot_Module_AutoConnect, Erebot_Module_PingReply.

Installing Erebot as a PHAR archive only involves a few steps:

1. Make sure your installation fulfills all of the prerequisites.

   Note: As all of Erebot's PHAR archives (core and modules) are digitally signed, you must make sure the OpenSSL extension is enabled on your PHP installation. Failure to do so will result in an error when trying to run Erebot's PHAR archive.

2. Download the PHAR archive for Erebot itself. You can grab the latest version from https://packages.erebot.net/get/Erebot-latest.phar. You MUST also download the public signature for the archive. The signature for the latest version is available at https://packages.erebot.net/get/Erebot-latest.phar.pubkey.

3. Create a directory named `modules` in the same folder as the PHAR.

4. Go to the `modules` directory and drop a copy of the following PHAR archives with their signature:
   - Files for the Erebot_Module_AutoConnect module:
     - Erebot_Module_AutoConnect-latest.phar
     - Erebot_Module_AutoConnect-latest.phar's signature
   - Files for the Erebot_Module_IrcConnector module:
     - Erebot_Module_IrcConnector-latest.phar
     - Erebot_Module_IrcConnector-latest.phar's signature
   - Files for the Erebot_Module_PingReply module:
     - Erebot_Module_PingReply-latest.phar
     - Erebot_Module_PingReply-latest.phar's signature

   Make sure you also read each component's documentation (especially the list of prerequisites).

   Note: You **MUST** copy both the PHAR archives and their signature in the `modules` directory. Otherwise, PHP will refuse to load those PHAR archives because it cannot check their origin and integrity.

5. Optionally, download additional PHAR archives with their signature to install other modules.

Your tree should now look like this:

- **Erebot/**
  - Erebot-latest.phar
  - Erebot-latest.phar.pubkey
  - **modules/**
    - Erebot_Module_AutoConnect-latest.phar
    - Erebot_Module_AutoConnect-latest.phar.pubkey
    - Erebot_Module_IrcConnector-latest.phar
    - Erebot_Module_IrcConnector-latest.phar.pubkey
    - Erebot_Module_PingReply-latest.phar
    - Erebot_Module_PingReply-latest.phar.pubkey
    - *eventually, additional PHAR archives with their signature*

Note: The whole installation process using PHAR archives can be automated using the following commands:

```
$ wget --no-check-certificate                                                  \
    https://packages.erebot.net/get/Erebot-latest.phar                          \
    https://packages.erebot.net/get/Erebot-latest.phar.pubkey                   \
    https://packages.erebot.net/get/Erebot_Module_AutoConnect-latest.phar       \
    https://packages.erebot.net/get/Erebot_Module_AutoConnect-latest.phar.pubkey \
    https://packages.erebot.net/get/Erebot_Module_IrcConnector-latest.phar      \
    https://packages.erebot.net/get/Erebot_Module_IrcConnector-latest.phar.pubkey \
    https://packages.erebot.net/get/Erebot_Module_PingReply-latest.phar         \
    https://packages.erebot.net/get/Erebot_Module_PingReply-latest.phar.pubkey
$ mkdir modules
$ mv Erebot_Module_*-latest.phar Erebot_Module_*-latest.phar.pubkey modules/
```

Once the PHAR archives have been retrieved, you may wish to change file permissions on `Erebot-latest.phar`, using this command:

```
$ chmod 0755 Erebot-latest.phar
```

This way, you may later launch Erebot simply by executing:

```
$ ./Erebot-latest.phar
```

Warning: Even though the command above should work on most installations, a few known problems may occur due to incompatibilities with certain PHP features and extensions. To avoid such issues, it is usually a good idea to check the following items:

- Make sure `detect_unicode` is set to `Off` in your `php.ini`. This is especially important on MacOS where this setting tends to be `On` for a default PHP installation.
- If you applied the Suhosin security patch to your PHP installation, make sure `phar` is listed in your `php.ini` under the `suhosin.executor.include.whitelist` directive.
- Please be aware of certain incompatibilities between the Phar extension and the ionCube Loader extension. To run Erebot from a PHAR archive, you will need to remove the following line from your `php.ini`:

```
zend_extension=/usr/lib/php5/20090626+lfs/ioncube_loader_lin_5.3.so
```

(the path and versions may be different for your installation).

Note: When run from a PHAR archive, Erebot will first try to determine whether all requirements needed to run the bot and its modules are respected. In case an error is displayed, follow the indications given in the error message and try running the bot again.

That's it! You may now read the section on *final steps* for a summary of what to do next.

2.2 Installation from source

First, make sure a git client is installed on your machine. Under Linux, **from a root shell**, run the command that most closely matches the tools provided by your distribution:

```
# For apt-based distributions such as Debian or Ubuntu
$ apt-get install git

# For yum-based distributions such as Fedora / RHEL (RedHat)
$ yum install git

# For urpmi-based distributions such as MES (Mandriva)
$ urpmi git

# For Zypper-based distributions such as SLES (SuSE)
$ zypper install git
```

Note: Windows users may be interested in installing Git for Windows to get an equivalent git client. Also, make sure that **`git`** is present on your account's `PATH`. If not, you'll have to replace **`git`** by the full path to `git.exe` on every invocation (eg. **`"C:\Program Files\Git\bin\git.exe" clone ...`**)

Also, make sure you have all the required dependencies installed as well. Now, retrieve the bot's code from the repository, using the following command:

```
$ git clone --recursive git://github.com/Erebot/Erebot.git
$ mkdir -p Erebot/vendor/ && cd Erebot/vendor/
$ git clone --recursive git://github.com/Erebot/Erebot_Module_IrcConnector.git
$ git clone --recursive git://github.com/Erebot/Erebot_Module_AutoConnect.git
$ git clone --recursive git://github.com/Erebot/Erebot_Module_PingReply.git
$ cd ..
```

Note: Linux users (especially Erebot developers) may prefer to create a separate checkout for each component and then use symbolic links to join them together, like this:

```
$ git clone --recursive git://github.com/Erebot/Erebot.git
$ git clone --recursive git://github.com/Erebot/Erebot_Module_IrcConnector.git
$ git clone --recursive git://github.com/Erebot/Erebot_Module_AutoConnect.git
$ git clone --recursive git://github.com/Erebot/Erebot_Module_PingReply.git
$ mkdir -p Erebot/vendor/ && cd Erebot/vendor/
$ ln -s ../../Erebot_Module_IrcConnector
$ ln -s ../../Erebot_Module_AutoConnect
$ ln -s ../../Erebot_Module_PingReply
$ cd ..
```

Optionally, you can compile the translation files for each component. However, this requires that gettext and phing be installed on your machine as well. See the documentation on Erebot's prerequisites for additional information on how to install these tools depending on your system.

Once you got those two up and running, the translation files can be compiled, assuming you're currently in Erebot's folder, using these commands:

```
$ phing
$ cd vendor/Erebot_Module_IrcConnector/
$ phing
$ cd ../Erebot_Module_AutoConnect/
$ phing
$ cd ../Erebot_Module_PingReply/
$ phing
$ cd ../../
```

Be sure to read the section on *final steps* for a summary of what to do next.

2.3 Final steps

Once Erebot (core files + a few modules) has been installed, you can write a configuration file for Erebot (usually named `Erebot.xml`).

When this is done, the bot can be started, assuming that PHP can be found in your `PATH` using one of the following commands. Exactly what command must be used depends on the installation method.

```
# For an installation using PHAR archives.
# Must be run from the folder in which Erebot was installed.
$ php ./Erebot-<version>.phar

# For an installation from sources or using Composer.
# Must be run from the folder in which Erebot was installed.
$ php ./scripts/Erebot
```

Let's call this command `%EREBOT%`.

In each case, the bot reacts to a few command-line options. Use the following command to get help on those options.

```
$ %EREBOT% --help
```

Note: For ease of use, Linux users may prefer to add the path where `Erebot-`*`version`*`.phar` or the **`Erebot`** script resides to their `PATH`. This way, the bot can be started simply by launching **`Erebot`** or `Erebot-`*`version`*`.phar` from the command-line or by double-clicking on them from a graphical file browser.

Note: Unfortunately for Windows users, there is no equivalent to the `PATH` trick noted above. However, it is possible to associate the `.phar` extension with PHP. This way, if Erebot was installed using PHAR archives, the bot can be started simply by double-clicking on `Erebot-`*`version`*`.phar`.

CHAPTER 3

Erebot's configuration

Erebot's configuration is stored in an XML file. This file is usually called "`Erebot.xml`", though you could name it otherwise and use the `Erebot -c` option when running Erebot to point it to your file.

This file is composed of a hierarchy of settings, with inner sections being able to inherit settings from outer sections.

The configuration is based on 3 structures:

- general settings
- logging configuration
- IRC-related settings

The general settings include things such as information on the current timezone, the locale (language) the bot should use to display messages in the console, etc.

The logging configuration is what defines what information the bot will print to the logs, how the log are organized (do we store them in a syslog, a database, or print them directly in the console) and how they appear (how they're formatted).

Last but not least, the rest of the configuration is dedicated to IRC, with information on what networks/servers the bot should contact, what modules it should enable, etc.

The rest of this page gives information on available options and possible values and is directly mapped to the actual hierarchy used in the XML configuration file.

Table of Contents

Note: The tags may be used in any order. Therefore, one could swap the general configuration for *<modules>* with the configuration for the *<logging>* subsystem in the tree above. You still need to maintain the hierarchy however.

Therefore, a <*channels*> or <*servers*> tag may only be a descendant of a <*network*> tag.

3.1 <configuration>

The <*configuration*> tag deals with settings related to the machine Erebot is running on more than to IRC itself.

The following table lists attributes of this tag with their role.

Table 3.1: Valid attributes for the <configuration> tag

Attribute	Default value	Required	Role
commands-prefix	n/a	**Yes**	The prefix used to identify commands adressed to the bot. Common values include: `!`, `'`, `@`, etc.
daemon	n/a	No	Whether to start the bot as a daemon (`True`) or not (`False`).
group	n/a	No	Once started, assume that group's identity (given as a GID or as a name).
language	n/a	**Yes**	The preferred locale to use, as an IETF language tag (eg. `en-US` or `fr-FR`). The usual Linux format for locales (`en_US`) is also supported.
pidfile	n/a	No	Store the bot's PID in this file.
timezone	n/a	**Yes**	This computer's current timezone, eg. `Europe/Paris`. [1]
user	n/a	No	Once started, assume that user's identity (given as a UID or as a name).
version	n/a	**Yes**	Must match the Erebot's version. It is currently used as a failsafe to prevent the bot from running with an outdated configuration file.

Note: The values of the `daemon`, `user`, `group` & `pidfile` options can be overriden from the command-line. The values given here only act as default ones in case the command line does not override them.

3.1.1 <logging>

The logging system used by Erebot is highly customizable. It uses the same kind of API as the Python logging module as it is actually a port of that module for PHP, hence its name PLOP (Python Logging On PHP).

It was developped as a subproject of Erebot and ships with its own documentation.

3.1.2 <modules>

Each of the <*configuration*>, <*network*>, <*server*> and <*channel*> tags may have a <*modules*> subtag to specify which modules should be made available at that level.

This tag is a simple container for zero or more <*module*> tags.

<module>

This tag defines a module that will be available at the current level (ie. either globally or for the current network/server/channel).

Settings for a module at one level will override settings for the same module at some higher level (hence, settings for a module in a <*channel*> section will replace settings defined at the <*network*> level). <*channel*> is considered as being at a lower level as <*server*> for the purposes of this mechanism.

[1] The list of supported timezones can be found on http://php.net/manual/en/timezones.php

You may choose to enable/disable a module at a particular level by setting its `active` attribute to `True` or `False` (respectively).

The following table lists attributes of this tag, their default value and their role.

Table 3.2: Valid attributes for the <module> tag

Attribute	Default value	Role
name	n/a	The name of the module to load/unload.
active	`True`	Indicates whether the module should be enabled at that level (`True`), or disabled (`False`).

A <module> tag may contain zero or more *<param>* tags to specify additional parameters the module should take into account (such as specific settings).

<param>

This tag can be used to define a parameter for a module. It has 2 (two) mandatory attributes, as described in the table below.

Table 3.3: Valid attributes for the <param> tag

Attribute	Default value	Role
name	n/a	The name of the parameter.
value	n/a	The value for that parameter. Different types of values are accepted. The precise type to use depends on the module and parameter. Read each module's documentation for more information.

A <param> tag may NOT contain any subtags.

3.1.3 <networks>

This tag is a simple container for zero or more *<network>*.

<network>

This tag represents an IRC network. The following table lists attributes of this tag with their role.

Table 3.4: Valid attributes for the <network> tag

Attribute	Default value	Role
name	n/a	The name of that IRC network.

The <network> tag **MUST** contain a *<servers>* subtag, used to describe IRC servers belonging to that IRC network.

It may contain a *<modules>* subtag to change the settings of a module for this IRC server.

It may also contain a *<channels>* subtag to change the settings of a module for some IRC channels on this network.

<servers>

This tag is a simple container for **one** or more *<server>*.

<server> This tag represents the configuration of an IRC server. The following table lists attributes of this tag with their role.

Table 3.5: Valid attributes for the <server> tag

Attribute	Default value	Role
url	n/a	Connection URLs to use to contact this IRC server.

The `url` attribute contains a series of connection URLs. A connection URL simply gives information on how to connect to a particular IRC server. A valid connection URL looks like this: `ircs://irc.iiens.net:7000/?verify_peer=0`

The scheme part may be either `irc` for plain text communications or `ircs` for IRC over SSL/TLS (encrypted communications). The host part indicates the IP address or hostname of the IRC server. The port part can be used to override the default port value for the given scheme.

By default, plain text IRC uses port 194 while IRC over SSL/TLS uses port 994. However, since both of these ports require root permissions on linux to launch a server, most IRC servers use different values like 6667 or 7000 for plain text communications and 6697 or 7002 for encrypted communications.

Last but not least, additional parameters may be used to control various aspects of the connection phase. At present time, these settings only affect encrypted connections (IRC over SSL/TLS), but they may be later extended to affect plain-text connections as well. The following table lists currently supported parameters:

Table 3.6: Valid parameters for connection URLs

Name	Valid values	Description
verify_peer	`0` or `1`	Check if the certificate really belongs to the target IRC server.
allow_self_signed	`0` or `1`	Consider self-signed certificates to be valid.
ciphers	a list of ciphers separated by colons	Acceptable ciphers to use to encrypt communications with the server.

See also http://php.net/manual/en/context.ssl.php for additional information on those settings.

You may also specify an HTTP or SOCKS 5 server through which the connection should be proxied by adding a proxy URL to the `url` attribute. Several proxies can be used by prepending their URLs to that attribute, separated by spaces:

```
<!-- Use an HTTP proxy with username/password authentication. -->
<server url="http://user:pass@proxy.example.com irc://irc.example.com"/>

<!-- Use a SOCKS 5 proxy with username/password authentication. -->
<server url="socks://user:pass@proxy.example.com irc://irc.example.com"/>

<!--
  Chain two proxies before connecting to the final IRC server.
  The first one is an HTTP proxy running on non-standard port 8080.
  The second one is a regular SOCKS proxy.
-->
<server url="http://http-proxy.example.com:8080/ socks://socks-proxy.example.com/ irc://irc.example.c
```

Warning: As of this writing, Erebot does not support older versions of the SOCKS protocol (namely, SOCKSv4 and its derivatives).

This tag may contain a *<modules>* subtag to change the settings of a module for this IRC server.

<channels>

This tag is a simple container for zero or more *<channel>* tags.

<channel> This tag represents the configuration of an IRC channel. The following table lists attributes of this tag with their role.

Table 3.7: Valid attributes for the <channel> tag.

Attribute	Default value	Role
name	n/a	The name of the IRC channel being configured.

This tag may contain a *<modules>* subtag to change the settings of a module for this IRC channel.

CHAPTER 4

Compatible modules

This page lists several modules which are known to be compatible with Erebot, divided into two categories:

1. *Supported modules*
2. *Third-party modules*

4.1 Supported modules

This section contains a list of supported modules:

- Admin
- AutoConnect
- AutoIdent
- AutoJoin
- AZ
- Countdown
- CtcpResponder
- Feeds
- GoF
- Helper
- IrcConnector
- IrcTracker
- LagChecker
- Math
- MiniSed
- PhpFilter
- PingReply
- RateLimiter
- Roulette
- ServerCapabilities
- Thiercelieux
- TriggerRegistry
- TV
- Uno
- WatchList
- WebGetter
- Wordlists

All supported modules can be found at https://github.com/Erebot/ (look for repositories whose name starts with *Erebot_Module_*). See each module's individual documentation for installation instructions and additional information.

4.2 Third-party modules

This sections contains a list of third-party modules:

No third-party module has been submitted yet.

Contact us at erebot@erebot.net if you would like your module to be listed here.

Warning: No support is provided by Erebot developpers for those third-party modules.

CHAPTER 5

Tutorials

This page lists various tutorials on how to configure Erebot to suit your needs:

5.1 How to interact with Erebot from an external process

This guide will show you how to setup Erebot so that an external process can interact with an IRC server through the bot. In the second part of this tutorial, we will also see how the logging system can be used to receive feedback from the bot for commands we sent to it.

Table of Contents

5.1.1 Sending commands through the bot

Erebot embeds a class called `Erebot_Prompt` that can be used to control the bot remotely using a UNIX socket. This can be used for example to build a web frontend for the bot. It might be used to build a complete IRC client too.

Warning: This feature only offers a one-way communication channel with the bot. That is, it can be used to send commands to the bot, but it cannot be used to see the actual responses to those commands.

Note: If you need bidirectional communications, you can combine this feature with Erebot's logging mechanism to intercept messages as the bot sends or receives them. See the section entitled « *Intercepting messages* » for more

information.

Warning: This feature is only available on platforms that implement UNIX sockets (especially, it is **not** available on Windows platforms).

Setting things up

Enabling the prompt is actually quite easy. All you need to do is add a service named "prompt" to your `defaults.xml` configuration file. That service will usually be an instance of the `Erebot_Prompt` class and should be passed the bot's service (named `bot`) as its first parameter. It also accepts a few parameters, listed in the following table.

Table 5.1: Parameters accepted by `\\Erebot\\Prompt` (in this order)

Parameter	Type	Description	Required?	Default value	Example value
$bot	object	Instance of the `bot` service.	Yes	N/A	N/A
$connector	string	Path to the UNIX socket to create.	No	"`/tmp/Erebot.sock`"	"`/var/lib/Erebot/control.sock`"
$group	string	UNIX group for the new socket.	No	Primary group of the user running the bot.	"`nogroup`"
$perms	integer	Permissions on the socket to create.		`0660` (`rw-rw----`)	`0666` (to allow any program to control Erebot—this is considered dangerous, avoid if possible).

Therefore, a potential configuration for the prompt in the `defaults.xml` configuration file may look like this:

```
<service id="prompt" class="\Erebot\Prompt">
    <argument type="service" id="bot"/>
    <argument>/var/lib/Erebot/control.sock</argument>
    <argument>nogroup</argument>
    <argument type="int">0666</argument>
</service>
```

Passing commands to Erebot

What you need to know

To send commands to Erebot, you need two pieces of information:

- The path to the UNIX socket that acts as Erebot's prompt.
- The name of the IRC network (as declared in Erebot's configuration file) to send the commands to.

Note: The latter is actually optional if you want to execute the command on all IRC networks (eg. an `AWAY` command before going to sleep), as we will see below.

A simple example

Once you have those information, open the UNIX socket using your favorite programming language.

Note: UNIX sockets can be opened from any language that supports them, including—but not limited to—Bash, Perl, PHP, Python, Java, etc.

You may now send commands using the following format:

```
<pattern> <command> <line ending>
```

where each token is described below:

<pattern> A pattern that will be used to match the network's name (as declared in Erebot's configuration file). You may use wildcard characters here (`?` to match 0 or exactly 1 character, `*` to match 0 or more characters). The simplest way to target a specific IRC network is to simply pass that network's name as the `<pattern>`.

<command> The IRC command you wish to send (eg. `AWAY :Gone to sleep`). Please refer to **RFC 2812** for information on valid commands.

<line ending> One of the 3 common line endings accepted by Erebot and noted below using C-style espace sequences:

- "`\r`" (Mac style)
- "`\n`" (Linux style)
- "`\r\n`" (Windows style)

Note: When looking for the connections targeted by a command, a case-insensitive full-line match is performed. This means that a pattern such as "`mynetwork`" and "`mynet*`" will match a network named "`MyNetwork`", but "`mynet`" won't.

Here is an example using the socat command from a cron task to make the bot quit the "`iiens`" IRC network every day at midnight:

```
1 # m h  dom mon dow   command
2   0 0  *   *   *     echo 'iiens QUIT :Time to sleep!' | socat - UNIX-SENDTO:/tmp/Erebot.sock
```

Targeting multiple IRC networks at once

As seen in the format above, a pattern matching the target IRC network's name is passed before the actual command. Hence, targeting multiple IRC networks at once is only a matter of using the right pattern. For example, if you have multiple connections to the same IRC network, named "`MyNetwork1`", "`MyNetwork2`", etc. you could easily send a command to all of these connections using "`MyNetwork*`" as the pattern.

Following the same logic, it is possible to send a command to **all** the servers the bot is currently connected to by using "`*`" as the pattern, since this will match any network, regardless of its name.

5.1.2 Intercepting messages

The technic described below makes it possible to intercept both incoming and outgoing messages. It is ideal if you're trying to build a frontend for Erebot because:

1. You can capture outgoing messages to get feedback on the actual commands being sent by the bot (keep in mind that modules may prevent certain commands from being sent for example).
2. You can capture incoming messages too, which means that you can process them using external tools if needed (eg. display them on your website).

Important: Even if you could easily process messages with an external tool then feed the results back to Erebot using the UNIX socket, it is often a lot more efficient to write a module for Erebot directly (using the assets provided by the PHP toolbox).

5.1.3 Troubleshooting

This paragraph lists the most common problems you may encounter while following this tutorial, as well as explanations as to why they appear and possible solutions or workarounds.

PHP Warning: stream_socket_server(): unable to connect to udg:///... (Unknown error) in .../Erebot/Prompt.php on line ...

Example:

```
PHP Warning:  stream_socket_server(): unable to connect to udg:///tmp/Erebot.sock (Unknown error) in
PHP Stack trace:
PHP   1. {main}() /home/clicky/Documents/Erebot/bin/Erebot:0
PHP   2. \Erebot\CLI::run() /home/clicky/Documents/Erebot/bin/Erebot:99
PHP   3. sfServiceContainer->__get() /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjecti
PHP   4. sfServiceContainerBuilder->getService() /var/local/buildbot/pear/php/SymfonyComponents/Deper
PHP   5. sfServiceContainerBuilder->createService() /var/local/buildbot/pear/php/SymfonyComponents/De
PHP   6. ReflectionClass->newInstanceArgs() /var/local/buildbot/pear/php/SymfonyComponents/Dependency
PHP   7. \Erebot\Prompt->__construct() /home/clicky/Documents/Erebot/src/Prompt.php:0
PHP   8. stream_socket_server() /home/clicky/Documents/Erebot/src/Prompt.php:44
```

Origins:

This error usually appears after the bot was stopped in a non-clean fashion (eg. after it has been killed). This is caused by a left-over UNIX socket created by the previous instance. You can fix the problem by manually removing the socket.

Solution:

Issue the following command (adapt the path depending on the content of the error message):

```
rm -f /tmp/Erebot.sock
```

PHP Fatal error: Uncaught exception 'Exception' with message 'Could not change group to '...' for '...'' in ...

Example:

```
PHP Fatal error:  Uncaught exception 'Exception' with message 'Could not change group to 'nogroup' fo
Stack trace:
#0 [internal function]: \Erebot\Prompt->__construct(Object(Erebot), '/tmp/Erebot.soc...', 'nogroup',
#1 /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjection/sfServiceContainerBuilder.php(2
#2 /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjection/sfServiceContainerBuilder.php(8
#3 /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjection/sfServiceContainer.php(276): si
```

```
#4 /home/clicky/Documents/Erebot/src/CLI.php(363): sfServiceContainer->__get('prompt')
#5 /home/clicky/Documents/Erebot/bin/Erebot(99): \Erebot\CLI::run()
#6 {main}
  thrown in /home/clicky/Documents/Erebot/src/Prompt.php on line 56
```

Origins:

Possible reasons for this error include:

- The given group name or GID does not exist.
- The current user is not the superuser (root) and is not a member of the given group (this is a limitation from the low-level chgrp system call). See also http://php.net/chgrp for more information.

Solution:

Make sure the given group exists and the user running the bot is a member of that group (or is the superuser).

CHAPTER 6

Contribute

So, you took interest in Erebot and would like to contribute back? This is the right page!

There are actually several ways by which would may contribute to the project:

- by reporting new issues (or asking for new features)
- by submitting *new modules*
- by forking the code and sending *pull requests*
- by running the tests suite (manually or by providing a *test machine*)
- by submitting *new translations* (or patches for existing ones)

Whichever one it is, you may also join our IRC channel to discuss issues, new ideas / feature requests and follow the bot's development.

6.1 New modules

If you wish to contribute new modules, you should probably start by reading our *development guides*. Once your code is ready, make sure to send an email to erebot@erebot.net with a link to it so that your module can be added to our list of *Third-party modules*.

6.2 Code/patch for Erebot's core and API

If you plan on sending patches (or pull requests), please read our documentation on the coding standard used by Erebot developers first. Your patch will have greater chances to be approved if it abides by that standard when you submit it.

To contribute a patch, you will need a GitHub account. Then you can simply:

- Fork the code to your own account.
- Create a new branch.
- Patch things up as much as you want.
- Create a pull request with your changes.

Once your pull request has been received, it will undergo a review process to decide whether it can be accepted as-is, needs more changes before having a chance to be accepted or is utterly rejected.

6.2.1 Buildslaves

A buildslave is a machine that runs continuous integration tasks for the project. Such tasks include running unit tests after every commit, building the online documentation, packaging the project, etc.

If you wish to contribute a machine for the project, this page explains how to do so.

Table of Contents

Foreword

Note: Even though a buildslave can do many things, we only discuss use of buildslaves to run unit tests in the context of this page. That's because we do not intend to make remote machines handle critical tasks such as packaging that involve digital signatures.

Warning: When running a buildslave, some PHP code will be run on your system without requiring any prior confirmation. This may represent a risk for your machine. Even though we do our best to prevent malicious code execution on the buildslaves, we still depend on external services (GitHub, Transifex, etc.) and we cannot guarantee that no malicious code will ever be injected in the repository.
We therefore strongly encourage you to take every necessary precaution **before** running the buildslave. This means that you should at least:

- Make sure your system is up-to-date regarding security patches.
- Run an antivirus with up-to-date virus signatures.
- Create a specific unprivileged account for the buildslave.
- Run the buildslave in a jail/chroot environment on platforms that support this feature.

By providing a buildslave, you agree that you understand and accept all of the risks mentionned above and that you are sole responsible for ensuring the security of your machine. If you disagree with that, please **do not** submit a request to be registered as a buildslave.

We already host one buildslave and a few virtual machines that each come with their own buildslave. At the time of this writing, Erebot is therefore tested against the following operating systems / distributions:

- Microsoft Windows XP SP3 i386 (virtual machine)
- CentOS 6.2 i386 (virtual machine)
- Debian 6.0 x86_64

Each buildslave runs with a variety of PHP versions (usually only one, but sometimes more). We use phpfarm to manage the different versions. Each version of PHP is tested separately and a buildslave can use up to 10 different PHP versions.

Since the buildmaster (the server that tells the buildslaves to test something) has no knowledge of what versions a given buildslave has, the buildslave must declare the versions it supports. To do so, code such as the following must be added at the very beginning of the buildslave's `buildbot.tac` file:

```
import os
os.environ['PHP1_PATH'] = '/home/foo/phpfarm/inst/php-5.2.17-debug/bin/:/home/foo/phpfarm/inst/php-5.
os.environ['PHP2_PATH'] = '/home/foo/phpfarm/inst/php-5.3.10-debug/bin/:/home/foo/phpfarm/inst/php-5.
os.environ['PHP3_PATH'] = '/home/foo/phpfarm/inst/php-5.4.0-debug/bin/:/home/foo/phpfarm/inst/php-5.4
```

```
os.environ['PHP1_DESC'] = '5.2.17-debug'
os.environ['PHP2_DESC'] = '5.3.10-debug'
os.environ['PHP3_DESC'] = '5.4.0-debug'
os.environ['PHP_MAIN'] = '3'
```

The `PHPx_PATH` environment variables specify additional directories to add to the `PATH` environment variable when using the PHP version with identifier `x`. You may specify multiple paths by using the appropriate separator for your operating system (eg. colon on Linux, semi-colon on Windows).

The `PHPx_DESC` lines specify a user-friendly description of the PHP version with identifier `x`.

Last but not least, `PHP_MAIN` specifies what is considered the "main" PHP version supported by the buildslave. When some PHP code must be run but we do not care what version of PHP is used, the version with that identifier will be used. In the example above, the main version is '3', which refers to PHP 5.4.0-debug.

Note: The versions may be numbered from 1 to 10 **with no gap in between**. Any gap in the numbering will result in the versions following the gap to not be tested at all.

If you want to test the code against more than 10 different versions, either run a separate buildslave on the same machine with additional versions or contact us on IRC or GitHub so that we increase the current limit.

Warning: At a minimum, you must define at least 3 variables (`PHP1_PATH`, `PHP1_DESC` and `PHP_MAIN`, where `PHP_MAIN` equals "1").
When adding a new version of PHP to test against, you must always specify both the `PHPx_PATH` and `PHPx_DESC` variables

Microsoft Windows

Linux

6.3 Test machines

So, you have some spare CPU cycles to contribute? We're glad to hear that!

The Erebot project uses Buildbot for its continuous integration. The process of setting up a test machine (more oftenly called a "buildslave" in Buildbot's terminology) is a bit tedious and deserves a page of its own.

6.4 Translations

The project uses the Transifex service to manage translations. All submissions or patches to translations should be submitted through Erebot's project page on Transifex.

To submit a new translation or a patch for an already existing translation, you will need a Transifex account. Then, apply for one of the translation teams or request the creation of a new team in case none currently exists for your language. As soon as you have joined one of the translation teams, you may proceed with your changes.

Just like for the code, translations undergo manual review. Once a translation has been formally reviewed, it will automatically be merged in the project alongside other translations.

Note: There is a tight integration between the code and the translations. The translations on Transifex are periodically synchronized with the messages in the repository, meaning that any change / new message in the code quickly appears

on the Transifex page.

This goes both ways: changes to the translations are automatically merged in the code's repository once they have been approved by one of your team's reviewers.

Note: It usually takes from 5 to 10 minutes after a translation has been approved before the changes are visible in the repository.

CHAPTER 7

Developer Zone

This pages contains links to various resources that developers may be interested in:

7.1 Coding standard

This page contains documentation on the coding standard used for Erebot's development. It takes inspiration from other such documents, namely:

- Drupal coding standards
- Zend Framework coding style
- The PEAR coding standards

Most of the information on this page is organized in the same way as the Drupal coding standards.

Table of Contents

7.1.1 Indenting and whitespace

Use an indent of 4 spaces, with no tabs.

Lines should have no trailing whitespace at the end.

Files should be formatted with \n as the line ending (Unix line endings), not \r\n (Windows line endings).

All text files should end in a single newline (\n). This avoids the verbose "\ No newline at end of file" patch warning and makes patches easier to read since it's clearer what is being changed when lines are added to the end of a file.

7.1.2 Operators

All binary operators (operators that come between two values), such as +, −, =, !=, ==, >, etc. should have a space before and after the operator, for readability.

For example, an assignment should be formatted as

```
$foo = $bar;
```

rather than

```
$foo=$bar;
```

As a special case for assigning operators, such as =, &=, <<=, etc. you may add several spaces before the operator when the code contains several lines of assignments (lining up the equal signs) so as to improve readability:

```
<?php
    $foo     = 42;
    $bar    &= $foo;
    $bar   <<= 2;
?>
```

Unary operators (operators that operate on only one value), such as ++, should not have a space between the operator and the variable or number they are operating on.

When using the ternary operator, add parentheses around the condition: `(...) ? ... : ...`.

Only wrap the ternary operator if the total length of the line exceeds the 80 chars limit. In that case, add a single newline before the `?` and `:` symbols, while adding enough spaces before them so as to line up the 3 (three) parts of the operator:

```
<?php
    $longVariableNameCausingWrapping =  ($config->overridesDefaultValues())
                                        ? $config->getOverrides()
                                        : $config->getDefaultValues();
?>
```

7.1.3 Casting

Put a single space between the (type) and the operand of a cast:

```
(int) $mynumber
```

7.1.4 Control structures

Control structures include `if`, `for`, `while`, `switch`, etc. Here is a sample `if` statement, since it is the most complicated of them:

```
if (condition1 || condition2) {
    action1;
}
else if (condition3 && condition4) {
    action2;
}
else {
    defaultaction;
}
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

Use `else if` instead of `elseif`.

You are strongly encouraged to always use curly braces even in situations where they are technically optional.Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

The only exception to the rule above is to follow the "return/fail early" principle where the action following a condition is a `return` or `throw` statement and no line is ever going to be added to the action part. More information on this principle and why you should follow it can be found in the following blog entries:

- http://vocamus.net/dave/?p=1421 (by a long time Mozilla contributor)
- http://saltybeagle.com/2011/06/fail-early/ (by Brett Bieber, one of the main contributors to Pyrus, the next generation PEAR installer)

For switch statements:

```
switch (condition) {
    case 1:
        action1;
        break;

    case 2:
        action2;
        // defaultaction must also be executed in this case.

    default:
        defaultaction;
}
```

Note: It is sometimes useful to write a case statement which falls through to the next case by not including a `break` or `return` within that case. To distinguish these cases from bugs, any case statement where `break` or `return` are omitted should contain a comment indicating that this is the intended behaviour.

For do-while statements:

```
do {
    actions;
} while ($condition);
```

Warning: Use of the alternative forms for these structures is prohibited. For example:

```
<?php
    // DON'T DO THIS
    while ($foo):
        ...
    endwhile;
?>
```

7.1.5 Line length and wrapping

The following rules apply to code:

- In general, all lines of code should not be longer than 80 chars.
- Long control structure conditions should be wrapped into multiple lines so as not to break the 80 chars rule.

  Whenever possible, try to prepare values related to the condition beforehand (storing them in temporary variables if necessary).

  So, instead of this:

```
    if ($something['with']['something']['else']['in']['here'] == mymodule_check_something($whatever[
        ...
    }
```

  use the following snippet:

```
    $here = $something['with']['something']['else']['in']['here'];
    if ($here == mymodule_check_something($whatever['else'])) {
```

```
    ...
}
```

When breaking a test composed of several conditions, wrap the conditions after the operator (`&&` or `||`) and indent the next line using 4 spaces so as to line up the conditions.

So, instead of this snippet:

```
if (isset($something['what']['ever']) && $something['what']['ever'] > $infinite && user_access('
    ...
}
```

use this one:

```
if (isset($something['what']['ever']) &&
    $something['what']['ever'] > $infinite &&
    user_access('galaxy')) {
    ...
}
```

- Control structure conditions should also **NOT** attempt to win the *Most Compact Condition In Least Lines Of Code Award*™:

```
// DON'T DO THIS!
if ((isset($key) && !empty($user->uid) && $key == $user->uid) || (isset($user->cache) ? $user->c
    ...
}
```

Instead, it is recommended practice to split out and prepare the conditions separately, which also permits documenting the underlying reasons for the conditions:

```
// Key is only valid if it matches the current user's ID, as otherwise other
// users could access any user's things.
$is_valid_user = (isset($key) && !empty($user->uid) && $key == $user->uid);

// IP must match the cache to prevent session spoofing.
$is_valid_cache = (isset($user->cache) ? $user->cache == ip_address() : FALSE);

// Alternatively, if the request query parameter is in the future, then it
// is always valid, because the galaxy will implode and collapse anyway.
$is_valid_query = $is_valid_cache || (isset($value) && $value >= time());

if ($is_valid_user || $is_valid_query) {
  ...
}
```

Note: This example is still a bit dense. Always consider and decide on your own whether people unfamiliar with your code will be able to make sense of the logic.

7.1.6 Function/method calls

Functions and methods should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon.

Here's an example:

```
$var = foo($bar, $baz, $qux);
```

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable (as documented in the section on *Operators*).

Warning: Call-time pass-by-reference is strictly prohibited. See the section on *function/method declarations* for the proper way to pass function arguments by-reference.

In the case of a block of related assignments, more space may be inserted to line up function calls and promote readability:

```
$short          = foo($bar);
$longVariable   = foo($baz);
```

Warning: For methods/functions defined by the core of PHP or any of its extension, (that is, anything that isn't userland-define), always respect the case given by the PHP manual. Even though PHP is case-insensitive for most identifiers, there are recurring propositions about turning it into a case-sensitive language for everything. Using the official case from the start makes the code forward-compatible if such a change is ever made.

7.1.7 Function/method declarations

Note: We recommend that you use classes instead of functions in your code, even if it means creating classes containing static methods only. The rationale behind this decision being that it avoids global scope name pollution.

Warning: Call-time pass-by-reference is strictly prohibited.

Always put the opening curly brace on a new line.

```
function funstuff_system($field)
{
    $system["description"] = t("This module inserts funny text into posts randomly.");
    return $system[$field];
}
```

Arguments with default values go at the end of the argument list.

Use type-hints whenever possible, but only if the type-hint is `array` or **refers to an interface**.

```
<?php
    // Wrong:
    function make_cat_speak(GarfieldTheCat $cat) {
      print $cat->meow();
    }

    // Correct:
    function make_cat_speak(FelineInterface $cat) {
      print $cat->meow();
    }
?>
```

For classes provided by PHP or one of its extensions (eg. DOMDocument), consider writing an interface for it and use that as a type-hint.

Using an interface instead of a class name in the type-hint makes it easier to use a class that provides the same features (the same API) through a different implementation. This is especially useful when unit testing the function.

When a function or method's arguments list exceeds the 80 chars limit, use a single newline after the opening parentheses, write each argument on a separate line and put the closing parentheses on a separate line too. Indent each argument's line by 4 (four) spaces and add extra spaces to line up the arguments' dollar sign whenever type-hints and/or references are used In this case, the closing parentheses and the opening curly brace that follows it should still be on separate lines:

```
function foobar(
    Foo_Interface                   $foo,
    FooBar_Converter_Interface      $converter,
                                   &$qux
)
{
    ...
}
```

Last but not least, always attempt to return a meaningful value from a function if one is appropriate. If no meaningful value exist, consider returning `NULL` or an empty array instead of `FALSE`.

The return value must not be enclosed in parentheses. This can hinder readability, in addition to breaking code if a method is later changed to return by reference.

For example:

```
function send_notificationWRONG(User_Interface $user, $message)
{
    if (!$user->valid()) {
        // WRONG:   makes it harder to distinguish an invalid user
        //          from a failure while sending the notification.
        return FALSE;
    }

    // WRONG:   will trigger a fatal error if the function
    //          is ever modified to return by reference.
    return (mail($user->getMail(), 'Notification', $message));
}

function send_notificationOK(User_Interface $user, $message)
{
    if (!$user->valid())
        return NULL;

    return mail($user->getMail(), 'Notification', $message);
}
```

Exceptions may also be used instead of returning `NULL`. Whether an exception should be raised or `NULL` / an empty array returned is left to the appreciation of developpers.

7.1.8 Class constructor calls

When calling class constructors with no arguments, always include parentheses:

```
$foo = new MyClassName();
```

This is to maintain consistency with constructors that have arguments:

```
$foo = new MyClassName($arg1, $arg2);
```

Note that if the class name is a variable, the variable will be evaluated first to get the class name, and then the constructor will be called.

Use the same syntax:

```
$bar = 'MyClassName';
$foo = new $bar();
$foo = new $bar($arg1, $arg2);
```

7.1.9 Arrays

Arrays should be formatted with a space separating each element (after the comma), and spaces around the => key association operator, if applicable:

```
$some_array = array('hello', 'world', 'foo' => 'bar');
```

Note that if the line declaring an array spans longer than 80 characters, each element should be broken into its own line, and indented one level. Extra spaces may be added before the => operator to increase readability:

```
$form['title'] = array(
    '#type'          => 'textfield',
    '#title'         => t('Title'),
    '#size'          => 60,
    '#maxlength'     => 128,
    '#description'   => t('The title of your node.'),
);
```

Note: Always add a comma at the end of the last array element. It helps prevent parsing errors if another element is placed at the end of the list later.

7.1.10 Quotes

Erebot does not have a hard standard for the use of single quotes vs. double quotes. Where possible, keep consistency within each module, and respect the personal style of other developers.

With that caveat in mind: single quote strings are known to be faster because the parser doesn't have to look for in-line variables. Their use is recommended except in two cases:

1. In-line variable usage, e.g. `<h2>$header</h2>`.
2. Translated strings where one can avoid escaping single quotes by enclosing the string in double quotes. One such string would be "He's a good person." This string would become 'He's a good person.' with single quotes. Such escaping may not be handled properly by .pot file generators for text translation, and it's also somewhat awkward to read.

For long chunks of texts, you may also heredoc/nowdoc strings, except when the text needs to be translated, because the current parser for translations does not pick them up.

7.1.11 String concatenations

We recommend that you always use a space between the dot and the concatenated parts to improve readability (we current ruleset does not enforce this rule though).

```php
<?php
    $string = 'Foo' . $bar;
    $string = $bar . 'foo';
    $string = bar() . 'foo';
    $string = 'foo' . 'bar';
?>
```

When you concatenate simple variables, you can use double quotes and add the variable inside; otherwise, use single quotes.

```php
$string = "Foo $bar";
```

When using the concatenating assignment operator `.=`, use a space on each side as with the assignment operator:

```php
<?php
    $string .= 'Foo';
    $string .= $bar;
    $string .= baz();
?>
```

7.1.12 Comments

Don't use Perl-style commands (`# Comment`). For comments that span several lines, we recommend that you use C++ comments (`/* Comment */`).

When using C++ comments, you may use asterisks ("stars") at the start of each line.

Warning: Use of comments such as `/** ... */` or `///` is reserved for API documentation purposes using Doxygen commands. You **MAY NOT** use them to explain the logic of your code. Use the regular forms `/* ... */` & `//` instead in such cases.

For example,

```php
<?php
    // Connects the bot to the default servers.
    $bot->connect();

    /* This is a very long comment about the purpose of the snippet
     * of code that goes right after this comment, so as to explain
     * what it does (in case this may not be easy to understand) as
     * well as how it is done (for example, to describe side-effects).
     */
    do_something_very_complex();

    # This kind of comments MUST NOT be used.
    oops();

    /**
     * \brief
     *      A well known pseudo-random number generator.
     *
     * This type of comments may only be used to describe the API,
     * using Doxygen commands.
     */
    class PRNG
    {
        // The next comment describes part of this class' API.
```

```
        /// Seed for the PRNG.
        const SEED = 4;

        /**
         * \brief
         *      Return a new random number.
         *
         * \retval int
         *      Some random number.
         */
        public static function getRandomNumber()
        {
            return self::SEED;
        }
    }
?>
```

7.1.13 Filesystem paths

Never use any OS-specific directory separator (eg. "/") directly to concatenate parts of a path together. Always use the `DIRECTORY_SEPARATOR` constant instead. It will take care of abstracting differences in the separator used by each OS for you.

7.1.14 Including code

Note: For code that is part of Erebot itself, you don't need to manually include any file as the autoloader will load the files on the fly whenever this is required.

Anywhere you are unconditionally including a class file, use `require_once()`. Anywhere you are conditionally including a class file (for example, factory methods), use `include_once()`. Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them—a file included with `require_once()` will not be included again by `include_once()`.

Note: `include_once()` and `require_once()` are statements and not functions. Having said that, we recommend that you always put parentheses around the file name to be included, even though this is not necessary from a technical point of view. This makes the coding style coherent with that of functions.

Never use relative paths when including code, always build an absolute path. You may use the `__FILE__` magical constant and the `dirname()` function to help you build such a path. See also the *conventions for filesystem paths* for more information.

Note: Even for external libraries, we recommend that you use the autoloader provided by those libraries if one is available instead of manually including their code.

7.1.15 PHP code tags

Always use :

```
<?php ?>
```

to delimit PHP code, not the shorthand,

```
<? ?>
```

or other exotic tags allowed by PHP:

```
<script language="php">
    // DON'T USE THIS TAG.
    //
    // Many tools do not even recognize this block
    // as containing some PHP code.
    //
    // Also, the "language" attribute on script tags
    // has been deprecated since HTML 4.01.
</script>

<%
    // DON'T USE THIS TAG EITHER.
    // This tag conflicts with the one used for ASP code.
%>
```

This is required for portability across differing operating systems and set-ups.

Warning: Never use a closing `?>` at the end of code files:

- Removing it eliminates the possibility for unwanted whitespace at the end of files which can cause strange outputs to the console.
- The closing delimiter at the end of a file is optional anyway.
- PHP.net itself removes the closing delimiter from the end of its files, so this can be seen as a "best practice."

7.1.16 Semicolons

The PHP language requires semicolons at the end of most lines, but allows them to be omitted at the end of code blocks.

Erebot coding standards require them, even at the end of code blocks. In particular, for one-line PHP blocks:

```
<?php
    print $tax; // YES
?>
<?php
    print $tax  // NO
?>
```

7.1.17 Example URLs

Use `example.com` as the domain for all example URLs, per **RFC 2606**. You may also refer to subdomains of this domain, eg. `irc.example.com`.

7.1.18 Naming Conventions (Functions, Constants, Global Variables, Classes, Files)

Files

Files containing classes should be named after the content of the last underscore (_) contained in the class name. If the class name does not contain any underscore, the file should be named after the class name as a whole.

Warning: You must **ALWAYS** use a separate file for each class or interface defined in your code.

The file should also be placed in a hierarchy of directories that is directly mapped from each segment of the class name obtained after splitting the class name on underscores and removing the last segment (name of the file itself).

The same holds for interfaces.

The following table shows how files should be arranged depending on the name of the class/interface they contain:

Table 7.1: Class/interface name to filesystem mapping

Class name	Path on filesystem
Erebot	*src*/Erebot.php
Erebot_Module_Foo	*src*/Erebot/Module/Foo.php
Erebot_Interface_I18n	*src*/Erebot/Interface/I18n.php

This convention is required to make Erebot's autoloader work.

Classes and interfaces

Classes should be named using "UpperCamelCase", a newline should be inserted before the `extends` and `implements` keywords and before the opening curly brace.

When a class implements several interfaces, add a single newline after each comma separating the interfaces; do not put any space before the comma. Add extra spaces to line up the class and interface names.

For example:

```
<?php
    abstract class  ConnectionPool
    extends         PDO
    implements      Countable,
                    IteratorAggregate
    {
        ...
    }
?>
```

Use underscores when you need to logically separate groups of classes. For example, all classes belonging to an Erebot module start with the prefix `Erebot_Module_`*`ModuleName`*. See also *Files* for implications.

For an interface, the text `Interface` should always appear in the interface's name, preferably at the end (eg. `Erebot_Module_FooInterface`). If you prefer to use a separate directory where all the interfaces are stored, this is also permitted (eg. `Erebot_Module_Foo_Interface_Generic`).

Class methods and properties

Class methods and properties should use "lowerCamelCase":

The use of `public` properties is strongly discouraged, as it allows for unwanted side effects. It also exposes implementation specific details, which in turn makes swapping out a class for another implementation (one of the key reasons to use objects) much harder. Properties should be considered internal to a class.

All methods and properties of classes must specify their visibility: `public`, `protected`, or `private`. The PHP 4-style `var` declaration must not be used.

The use of `private` class methods and properties should be avoided—use `protected` instead, so that another class could extend your class and change the behaviour of a method if necessary (eg. for unit testing purposes).

Use an underscore prefix for `protected` and `private` methods and properties so as to make them easily identifiable.

You may use extra spaces before a property's name to line up all properties of a class.

The following snippet summarizes all of those rules:

```
<?php
    class Foo
    {
        public      $statementCounter;
        protected   $_statement;
        private     $_lastStatement;
    }
?>
```

For methods that are `static`/`abstract`/`final`, PHP allows use of the keywords in any order. We do not impose any order either, except that the visibility specifier (`public`, `protected` or `private`) should always be the last keyword.

For example, the snippet below defines four methods. Only the first 2 (two) forms are accepted in Erebot, with the first form being the preferred one (`final` or `abstract` before `static`):

```
<?php
    class Foo
    {
        // This is valid PHP code, and it is accepted in Erebot.
        final static public function foo()
        {
            ...
        }

        // This is valid PHP code, and it is also accepted in Erebot.
        static final public function bar()
        {
            ...
        }

        // This is valid PHP code, but it is forbidden in Erebot's code.
        final public static function baz()
        {
            ...
        }

        // This is valid PHP code, but it is forbidden in Erebot's code.
        public static final function qux()
        {
            ...
        }
    }
```

Class constructors

Always use the `__construct()` method to define a class constructor. Do not use the old PHP 4 convention where the constructor was named after the class:

```
<?php
    // NO
    class Bar
    {
        public function Bar()
        {
            ...
        }
    }

    // YES
    class Baz
    {
        public function __construct()
        {
            ...
        }
    }
?>
```

Functions

Functions should be named using lowercase, and words should be separated with an underscore. Functions should also have the module's name as a prefix, to avoid name collisions between modules.

Constants

- Constants should always be all-uppercase, with underscores to separate words. (This includes pre-defined PHP constants like `TRUE`, `FALSE`, and `NULL`.)
- Global constants defined by modules should also have their names prefixed by an uppercase spelling of the module that defines them.

  Note: Whenever possible, use class constants instead of global constants to avoid global naming space pollution.

- Global constants should be defined using the `const` PHP language keyword (instead of `define()`), for performance reasons:

```
    <?php
        /**
        * Indicates that the item should be removed
        * at the next general cache wipe.
        */
        const CACHE_TEMPORARY = -1;
    ?>
```

  Note: The `const` keyword does not work with PHP expressions. `define()` should still be used when defining a constant conditionally or with a non-literal value:

```
<?php
    if (!defined('MAINTENANCE_MODE')) {
        define('MAINTENANCE_MODE', 'error');
    }
?>
```

Global variables

Global variables are strictly forbidden in Erebot and any of its modules; this is non-negociable.

7.1.19 Check your code

To check that your code complies with these standards, install the following PEAR packages on your machine:

- pear.php.net/PHP_CodeSniffer
- pear.phing.info/Phing

If you are creating your own module, write a `build.xml` file if you haven't done so yet. You can start with a copy of Erebot_Module_Skeleton's own build.xml file. You'll also want to make sure that your module follows the same layout as official Erebot modules and that you added Erebot_Buildenv as a git submodule to your module.

Now, go to the root directory of the component and run:

```
phing qa_codesniffer
```

This will check your code against the standards described here.

You may also be interested in *installing other PEAR packages* related to QA (Quality Assurance), and then running the full QA test suite with:

```
phing qa
```

7.2 Internationalization

Table of Contents

7.2.1 A practical example

Supporting I18N in your module is very simple. All you need is an instance of a translator (an instance of an object that implements Erebot_Interface_I18n) for the module. A translator for the module is automatically created along

instances of your module.

Once equipped with a translator, just call its `gettext()` method, passing the string to translate to it. Therefore, translating a string is as simple as the following snippet:

```
<?php
    $translator = $this->getTranslator($chan);
    $translator->gettext('This text will be translated');
?>
```

Module developpers may also be interested in the other methods provided by the translator object, presented in the API documentation for that class.

7.2.2 Managing translations

In the previous section, we saw how to integrate strings that will be translated. So... how does Erebot finds out the correct translation?

There is another special file in `data/i18n/module.po` where *module* is the name of your module (eg. `Erebot_Module_XYZ`). This file uses the gettext format and lists all messages marked as requiring a translation (as extracted from your source code when running `phing i18n`).

Also, every Erebot module contains a set of translations in `data/i18n/locale/LC_MESSAGES/module.po`, where *locale* is some locale identifier, expressed using the following format: `xx_YY` (where `xx` is the ISO 639-1 code for the language [1] and `yy` is the code for the country, eg. `en_US`) and *module* is the name of your module (eg. `Erebot_Module_XYZ`). Those files use the same format as the previous file and provide the translations for the messages listed in `data/i18n/module.po`.

To ease management of the translations, and especially the PO files (also called "catalogs"), a few tools are provided by Erebot as phing targets. These tools are discussed below.

Extracting strings marked for translation

The `extract_messages` target can be used to parse the code of your module and extract strings marked for translation. This will write out every string marked for translation into `data/i18n/module.po`. Example:

```
$ phing extract_messages
```

Adding translations for a new locale

Translations for a new locale can be added by using the `init_catalog` target and passing a `locale` parameter, like so:

```
# Creates a new translation catalog for the "de_DE" locale (german).
$ phing init_catalog -Dlocale=de_DE
```

Updating existing catalogs

Updating the catalogs is quite simple, just use the `update_catalog` target:

```
$ phing update_catalog
```

[1] See http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

Compiling the catalogs

Last but not least, the catalog files cannot be used directly by the bot. You first need to compile them using the `compile_catalog` phing target:

```
$ phing compile_catalog
```

This will generate MO files for the miscellaneous PO files described above.

7.2.3 Plurals

Correct pluralization of sentences is a big challenge when dealing with i18n.

Warning: Even though the gettext family of tools has some (incomplete, at least from my point of view) support for plurals, the original feature from gettext is not used by Erebot.

Erebot handles plurals in an elegant way, using a special set of markup in the styling API. Readers may be interested in the documentation on styling for more information on plurals support.

7.3 Formatting

This page is only meant to guide you through the use of formatting codes with Erebot, it is not meant as a complete documentation of how styles work with IRC. If you want more, ask your favourite search engine ;-)

Erebot provides two ways to format messages. Both methods are described here.

Table of Contents

Please note that using raw codes (method 1) is considered bad practice. Advanced formatting (method 2) should be used in new code.

7.3.1 Method #1 : raw codes

Warning: This method is now deprecated and should not be used in new code. This is because messages written using this method are very hard to internationalize. Please use the *second method* instead.

The formatting API provides constants for the raw control codes that make up styles. There are also constants for colors, see the source code for details.

For example:

```
<?php
    $user = "Foobar";

    // Using pseudo-HTML, this is the same as:
    // "<b>Hi <u>$user</u></b>"
    $message =  Erebot_Styling::CODE_BOLD.
                    "Hi ".
                    Erebot_Styling::CODE_UNDERLINE.
                        $user.
                    Erebot_Styling::CODE_UNDERLINE.
                Erebot_Styling::CODE_BOLD;
    print $message . PHP_EOL;
?>
```

The example above would display `Hi Foobar`. The message would be displayed in bold, with the user's nickname underlined when sent to an IRC server.

7.3.2 Method #2 : "advanced formatting"

This method aims at separating the process of designing the format string from the process of actually rendering it (producing the string that would be sent to an IRC server).

It uses an XML syntax to design the format string and a simple API to do the rendering. This syntax is generally more verbose than the previous one, but is also a lot easier to use and read. Moreover, each message is validated (using a RelaxNG schema [1]), making it impossible to build invalid templates.

Designing the format string

The format string contains XML tags which are recognized by the bot and allow you to achieve many different formatting combinations.

Currently, the following tags are available:

- `<b>` = bold
- `<u>` = underline
- `<color>` = change the (foreground and/or background) color of the text. This tag recognizes two optional attributes:
  - `fg` (optional) = changes the foreground color. The color's name should be one of the `COLOR_` constants defined in the styling class, with the `COLOR_` prefix stripped (eg. `red`). The color's name is case-insensitive.
  - `bg` (optional) = changes the background color. See the description of `fg` for valid values.

Note: At least one of the `fg` or `bg` must be provided, otherwise the message will be rejected as invalid.

- `<for>` = loop other an array to format its content. This tag recognizes 2 required attributes and some optional ones:
  - `from` (required) = name of the variable which contains the array to format.
  - `item` (required) = a variable which will be created to store each value in the array (in turn).

[1] For more information on RelaxNG schemae, see http://relaxng.org/.

  - `key` (optional) = a variable which will be created to store the key for that value (useful for associative arrays like in our example above).
  - `separator` (optional) = a separator to add between all entries in the array, except for the last two. Defaults to a comma followed by a single space.
  - `sep` (optional) = alias for `separator`.
  - `last_separator` (optional) = a separator to add between the last 2 entries of the array. If no `separator` attribute has been set, defaults to an ampersand between two spaces. Otherwise, defaults to the value of the `separator` attribute.
  - `last_sep` (optional) = alias for `last_separator`.

- `<var>` = insert the value of the given variable at this point. The value will be rendered in a locale-dependent way, depending on the *type of variable* used. This tag accepts one attribute:
  - name (required) = variable to insert. See *template variables* below for the various syntaxes supported by this attribute.
- `<plural>` = use the correct plural form for that sentence. This tag has a required attribute called `var` that is used to determine the correct plural form to use. See *template variables* below for the various syntaxes supported by this attribute.

  The content of this attribute should evaluate to an integer. Depending on the locale in use and this number, the appropriate plural form will be selected from a set of possibilities (cases).

  A `<plural>` tag contains one or more `<case>` subtags. Each `<case>` contains some inline text and comes with a required `form` attribute indicating when this text should be used [2].

  You **MUST** add a `<case>` subtag with the special form called `other`. This special form will be used when no specific rule applies for this word's plural.

Warning: If you're used to gettext's syntax for plurals (using a predicate and a fixed array of translations), you'll notice the format used here is much more flexible, as it enables one to write something such as:

```
There is/are <x> girl(s) and <y> boy(s) in this classroom.
```

using the *correct form for each word* (noun or verb), while gettext would require you to either split the text in multiple sentences or define a complicated predicate to retrieve the correct plural.
Also, please note that although gettext is used to store translations, the plural handling mechanism from gettext is never used by Erebot (ie. Erebot never calls `ngettext` or its variants). Instead, each message embeds both the singular and plural forms and an algorithm is used at runtime to decide which of the forms should be used.

Note: See also the documentation on the formatting API for more information.

Strong typing

Each variable in a template has an associated type. The following classes are available by default to represent some of the most common types:

`Erebot_Styling_Integer` Represents an integer.

```
<?php
    $formatter = new Erebot_Styling($translator);
    $source = '<var name="leet"/>';
```

[2] The page at http://unicode.org/cldr/data/charts/supplemental/language_plural_rules.html lists all available forms.

```
        $vars = array('leet' => new Erebot_Styling_Integer(1337));

        // This may be rendered as "1 337",
        // depending on the translator's locale.
        echo $formatter->_($source, $vars) . PHP_EOL;
    ?>
```

Erebot_Styling_String Represents a string. The value will be passed as is.

```
    <?php
        $formatter = new Erebot_Styling($translator);
        $source = '<var name="name"/>';
        $vars = array('name' => new Erebot_Styling_String('Clicky'));
        echo $formatter->_($source, $vars) . PHP_EOL;
    ?>
```

Erebot_Styling_Float Represents a floating-point value.

```
    <?php
        $formatter = new Erebot_Styling($translator);
        $source = '<var name="avg"/>';
        $vars = array('avg' => new Erebot_Styling_Float(1234.56));

        // This would be rendered as "1 234,56" in french.
        echo $formatter->_($source, $vars) . PHP_EOL;
    ?>
```

Erebot_Styling_Currency Represents a monetary value expressed in some currency.

```
    <?php
        $formatter = new Erebot_Styling($translator);
        $source = '<var name="price"/>';

        // Note: the currency can be passed as an additional parameter.
        // If omitted, the currency from the locale configured in the
        // $transator is used.
        $vars = array('price' => new Erebot_Styling_Currency(1234.567, 'EUR'));

        // This would be rendered as "€1,234.57" for US english.
        // Note that monetary values are rounded to two places.
        echo $formatter->_($source, $vars) . PHP_EOL;
    ?>
```

Erebot_Styling_DateTime Represents a date and/or time. Some extra values (passed as additional parameters to this class) are necessary to represent such data. Thus, the arguments for this class' constructor are:

- `$value`

  Either a DateTime object, an integer representing some Unix timestamp (seconds since Epoch, UTC) or an array using the same format as what is output by the localtime() PHP function.

  Note: DateTime objects are only supported since PHP 5.3.4, you should not rely on them in code intended to be backward compatible.

- `$datetype`

  One of `IntlDateFormatter::NONE`, `IntlDateFormatter::FULL`, `IntlDateFormatter::LONG`, `IntlDateFormatter::MEDIUM` or

`IntlDateFormatter::SHORT` [3]. This indicates how the date part of the value will be represented.

- `$timetype`

  One of `IntlDateFormatter::NONE`, `IntlDateFormatter::FULL`, `IntlDateFormatter::LONG`, `IntlDateFormatter::MEDIUM` or `IntlDateFormatter::SHORT`. This indicates how the time part of the value will be represented.

- `$timezone`

  A timezone identifier (such as "Europe/Paris"). This value is ignored when a Unix timestamp is passed as the `$value`.

```
<?php
    $formatter = new Erebot_Styling($translator);
    $source = '<var name="now"/>';
    $vars = array(
        'now' => new Erebot_Styling_DateTime(
            time(),
            IntlDateFormatter::FULL,
            IntlDateFormatter::FULL
        )
    );

    // In US English, this may be rendered like this:
    // "Wednesday, December 31, 1969 4:00:00 PM PT".
    echo $formatter->_($source, $vars) . PHP_EOL;
?>
```

`Erebot_Styling_Duration` Represents a duration in spelled out form, with a precision up to the seconds.

```
<?php
    $formatter = new Erebot_Styling($translator);
    $source = '<var name="duration"/>';
    $vars = array('duration' => new Erebot_Styling_Duration(1389722));

    // This would be rendered as:
    // "2 weeks, 2 days, 2 hours, 2 minutes, 2 seconds" in english.
    echo $formatter->_($source, $vars) . PHP_EOL;
?>
```

Tip: If you need to represent a value without any modification, pass it as a string or wrap it in an instance of `Erebot_Styling_String`.

Note: For basic scalar types (integer, string or float), the API will wrap the value automatically for you using the appropriate class (`Erebot_Styling_Integer`, `Erebot_Styling_String` or `Erebot_Styling_Float`, respectively). Arrays do not need to be wrapped in any class (but their values do!).

You may change the default classes used to wrap scalar types for a specific template using the `setClass()` method, eg:

```
<?php
    $translator = new Erebot_I18n();
    $tpl = new Erebot_Styling($translator);
```

[3] See http://php.net/class.intldateformatter.php for the meaning of each one of these constants.

```
    // Change the classes used to wrap basic scalar types.
    $tpl->setClass('int',       'Custom_Int_Wrapper');
    $tpl->setClass('string',    'Custom_String_Wrapper');
    $tpl->setClass('float',     'Custom_Float_Wrapper');

    // Use $tpl as we'd normally do.
?>
```

Template variables

When referencing a variable from a template using the `<var name="..."/>` or `<plural var="..."/>` tags, various syntaxes are available.

Hence, `...` may actually contain:

- Actual variable passed to the template, eg. `<var name="foo"/>`.
- The sum or difference between two integer or floating-point values, eg. `<var name="foo+bar"/>` or `<var name="foo-bar">`. Both types may be combined together (so, "foo" may refer to an integer, while "bar" refers to a floating-point value).

  You may use litteral integer or floating-point values as well, eg. `<var name="years-18"/>` or `<var name="century+1"/>`.

  Tip: As a special bonus, you may also use the add operator (+) to append the values of one array to another using `array_merge`. The original arrays are left intact when this feature is used.

  Warning: Any attempt to add or subtract values from incompatible types (eg. adding the value of an integer to a string) will result in an exception being thrown. In particular, subtracting one array from another is not supported yet.

  Warning: There is currently no plan to support the multiply (*) or divide (/) operators.

- Parenthesized expressions, eg. `<var name="totalCards-(nbCards+1)"/>`.
- The number of elements in an array passed to the template, using the "count operator" (#), eg. `<var name="#scores"/>`.

  Note: The count operator as higher precedence on the add/subtract operators, meaning that it is applied **before** any addition/substraction, unless parenthesis are used to override this.

  Warning: Use of the count operator on any other type may lead to unpredictable results.

- Whitespace (spaces or tabs), eg. `<var name="boys + girls"/>`. Such whitespace is ignored while processing the variable.

  Note: Due to limitations in the XML syntax, is it not possible to use newlines as whitespace.

- Any combination of the previous syntaxes, eg. `<var name=" # ( boys + girls ) "/>` where `boys` and `girls` both refer to arrays.

Warning: Please keep in mind that variable names are case-sensitive. Any attempt to use an undefined variable in a template will result in an exception.

Using templates in your code

Once the format string has been designed, you (as a programmer, not as a designer) must add a few lines in your code in order to use it.

This is usually done with the following steps:

1. Create an instance of Erebot_Styling by passing a translator object (an object implementing the Erebot_Interface_I18n interface) to its constructor. This is the creation step, where a formatter is created and bound to a translator.
2. Prepare the values (either scalar types, objects implementing the Erebot_Interface_Styling_Variable interface or arrays made of scalar types/objects) that will be used in the template. This is the preparation step, where everything is setup for the final step.

   Note: Variable names may only contain alphanumeric characters or the underscore (_) and dot (.) characters.

   Warning: While designing the template, keep in mind that variable names are case-sensitive.

3. Render the template (with `$fmt->render()` or `$fmt->_()`) and use the result of that process in your code (eg. send it to an IRC channel). This is the rendering step.

```php
<?php
    // The source for a template meant to display
    // the scores of each player in a fictitious game.
    $source =   '<b>Scores</b>: '.
                '<for item="score" key="nick" from="scores" '.
                    'separator=", " last_separator=" &amp;amp; ">'.
                    '<b>'.
                        '<u>'.
                            '<color fg="green">'.
                                '<var name="nick"/>'.
                            '</color>'.
                        '</u>'.
                        ': <var name="score"/>'.
                    '</b>'.
                '</for>';

    // Step 1:
    // Create a new translator and a new template from it.
    // By default, the locale for the translator is "en_US".
    $translator = new Erebot_I18n();
    $formatter  = new Erebot_Styling($translator);

    // Step 2:
    // Prepare some variables for the template.
    $vars = array(
        'scores' => array(
            'Foo' => 42,
            'Bar' => 23,
            'Baz' => 16,
```

```
                'Qux' => 15,
                'Toto' => 8,
                'Tata' => 4,
            ),
        );

        // Step 3:
        // Render the template with the given scores.
        //
        // This results in something like:
        // "Scores: Foo: 42, Bar: 23, Baz: 16, Qux: 15, Toto: 8 & Tata: 4"
        // with most of the words represented in bold
        // and the nicknames in green and underlined.
        //
        // Note: since we used "_()" to render the template,
        //       a translation is automatically selected (if available).
        echo $formatter->_($source, array('scores' => $scores)) . PHP_EOL;
    ?>
```

Here, `$source` has been split over many lines to make it easier to figure out how the final message will look like. The template could actually be written in a much more compact way.

You do not need to wrap your template (`$source`) in XML tags manually, the bot already adds an enclosing tag automatically for you.

Also, the format string could be retrieved from anywhere:

- an array in a PHP script,
- an external process (eg. a database),
- a translation catalog (MO file),
- etc.

We prefer to have customizable format strings in a translation catalog, as this gives more control to translators over the result and it is a format they are used to working with.

Plurals

Plurals are handled gracefully by Erebot using the `<plural>` and `<case>` tags.

Taking the sentence from earlier as an example:

```
There is/are <x> girl(s) and <y> boy(s) in this classroom.
```

The equivalent as a template would be:

```
<?php

    $msg = 'There '.
            '<plural var="#(girls+boys)"/>'.
                '<case form="one">is</case>'.
                '<case form="other">are</case>'.
            '</plural> '.
            '<plural var="girls"/>'.
                '<case form="one">one girl</case>'.
                '<case form="other"><var name="girls"/> girls</case>'.
            '</plural> '.
            'and '.
```

```
            '<plural var="boys"/>'.
                '<case form="one">one boy</case>'.
                '<case form="other"><var name="boys"/> boys</case>'.
            '</plural> '.
            'in this classroom';

    $formatter = new Erebot_Styling(new Erebot_I18n());

    // Displays "There is one girl and 0 boys in this classroom".
    echo $formatter->_($msg, array('girls' => 1, 'boys' => 0)) . PHP_EOL;

    // Displays "There are 2 girls and one boy in this classroom".
    echo $formatter->_($msg, array('girls' => 2, 'boys' => 1)) . PHP_EOL;

    // Displays "There are one girl and 2 boys in this classroom".
    echo $formatter->_($msg, array('girls' => 1, 'boys' => 2)) . PHP_EOL;
?>
```

Notice how we represented the actual counts using either a spelled out form ("one girl" / "one boy") or an actual number ("2 girls" / "2 boys"), simply by specifying different words for the different `<cases>`.

You'll also notice that this string is electable for Internationalization. Translators have full control over the template used to render the sentence and could easily adapt it to the plural rules used in their country.

Note: There are often many different ways to represent the same message using templates. Here, we grouped words that were affected by the same variable together. Once again, **translators are the ones in charge** here. This is very important because they know better than you how the sentence should look like in their language.

Further reading

The documentation on the formatting API always reflects the latest features implemented, while this page may sometime fall a little behind in what it showcases (please open a ticket if you notice any discrepancy!).

7.4 Writing a new module

This page acts like a guide for those who may be interested in writing a new module for Erebot. It assumes basic knowledge of some of the features provided by Erebot for developers (such as the styling features and i18n features).

Table of Contents

7.4.1 General structure

An Erebot module is a PHP class that extends `Erebot_Module_Base`. As such, it must have at least two methods (declared *abstract* in `Erebot_Module_Base`):

- `_reload()` is called when the module is (re)loaded with some flags giving more information about what must be (re)loaded. The flags are a bitwise-OR combination of the `RELOAD_*` constants found in `Erebot_Module_Base`.
- `_unload()` is called when the module is unloaded. Its purpose is to free any resource that may have been allocated by `_reload()`, save the current state elsewhere, etc.

Note: When a module is reloaded, only `_reload()` is called. The only time `_unload()` is ever called is when the module is being completely unloaded (usually, right before the bot exits).

7.4.2 Helping users

Note: Adding an help method to your module is totally optional, but it is considered good practice as it provides some way for users to request help on your new module and its commands without having to read some online manual.

To provide help for your module, all you need is a method that handles help requests. The name of that method does not matter (though this method is called `getHelp()` in all modules that ship with Erebot).

When someone requests help on a module or command, the help methods are looked up in order to find one that will acknowledge the request (see below). This may result in one or more help methods being called to handle the request.

The help method **must** use the following signature.

```
public function getHelp(
    Erebot_Interface_Event_Base_TextMessage $event,
    Erebot_Interface_TextWrapper            $words
)
```

This method is responsible for either acknowledging the help request (by returning `TRUE`) or ignoring it (by returning `FALSE` or by not returning anything at all). If your method chooses to ignore the help request, the next help method in line will be called with the same parameters, until either a method acknowledges the request or there are no more help methods to try.

`$event` will contain the original request as an event. This will either be an event that implements the `Erebot_Interface_Event_Base_Private` interface if the request was sent as a private query, or an event implementing the `Erebot_Interface_Event_Base_Chan` interface if it came from an IRC channel.

`$words` contains the content of the request (derived from the text in the original request in `$event`), wrapped to make it easier to look at individual words.

Now, there are two types of requests:

- Requests for help on the module itself (`!help Foo`). In that case, `$words` will contain only one word: the name of the module itself inside the `\\Erebot\\Module` namespace |–| `Foo` in this case.
- Requests for help on a command/topic (`!help foo`, `!help foo bar...`). In that case, `$words` will contain 2 or more words:
  - The name of the current module.
  - The name of the command (`foo`).

- Any additional parameters (`bar...`).

You can find out which type of request is in use by simply counting the number of words in `$words`, which is very easy as the wrapper implements the `Countable` interface:

```
// If it's 1, it is a request for help on the module itself.
// Otherwise, it's a request for help on some command/topic.
$nbWords = count($words);
```

Warning: Erebot has now way (yet) to know what module provides a given command/topic, so for such help requests, it calls every module's help method with the request until one acknowledges it.
This means that your help method may receive requests about commands or topics it knows nothing about. You **must** ignore such requests (by returning `FALSE` or nothing at all) and you **must not** send a message indicating an error in the request to the user.

The listing below shows an example of a very simple help method for an imaginary module:

```
public function getHelp(
    Erebot_Interface_Event_Base_TextMessage $event,
    Erebot_Interface_TextWrapper            $words
)
{
    if ($event instanceof Erebot_Interface_Event_Base_Private) {
        $target = $event->getSource();
        $chan   = NULL;
    }
    else
        $target = $chan = $event->getChan();

    $fmt        = $this->getFormatter($chan);
    $moduleName = strtolower(get_class());
    $nbArgs     = count($words);

    // Help request on the module itself.
    if ($nbArgs == 1 && $words[0] == $moduleName) {
        $msg = $fmt->_('This is an <b>imaginary</b> module.');

        // We send the message back to where the request came from:
        // in a private query or an IRC channel.
        $this->sendMessage($target, $msg);
        return TRUE;
    }

    // This module does not care about other help requests.
    // So we don't return anything here. This is the same
    // as if "return;" or "return NULL;" had been used.
}
```

Note: We used the `getFormatter()` method here to be able to format the help message (to make "imaginary" appear in bold in the output). We also used the formatter's `_()` method to mark the message for translating. This is the recommended practice.

Once the code for your help method is ready, you have to tell Erebot about it by using the `registerHelpMethod()` method inside your module's `reload()` method. You must call `registerHelpMethod()` with an object implementing the `\\Erebot\\Interface\\Callable` interface and referring to your method.

This can be done using the following snippet:

```
// First, we retrieve the factory to use to produce instances
// implementing "\Erebot\Interface\Callable".
$cls = $this->getFactory('!Callable');

// Next, we register our help method (here, the getHelp() method
// from the current object) by wrapping a callback-compatible
// value referring to it in a new callable object.
$this->registerHelpMethod(new $cls(array($this, 'getHelp')));
```

Alternatively, you may mark your module as implementing the **:api:'\\Erebot\\Interface\\HelpEnabled'** interface. In that case, the bot will automatically register the module's `getHelp()` method as the help method.

7.4.3 Frequently Asked Questions

This sections contains random questions about modules' development.

What features can I use in a new module?

You can use any of the many features provided by the PHP language. This includes things such as sockets, databases, etc.

Are there patterns I should avoid?

Even though you can do pretty much anything you want in a module, you should avoid long running tasks such as downloading a big file from a remote server.

The reason is simple: PHP does not support multithreading [1], so while a long running task is being executed, the rest of the bot is literally stopped. This includes other modules responsible for keeping the connection alive (`\\Erebot\\Module\\PingReply`). Hence, running a long task in your module may result in the bot being disconnected from IRC servers with a "Ping timeout" error.

[1] This is not entirely true anymore, as there is now an extension that brings the power of pthreads to PHP. Anyway, PHP does not natively support them and the extension has a few issues of its own. See https://github.com/krakjoe/pthreads for more information.

CHAPTER 8

Licenses

Table of Contents

8.1 Erebot

The documentation (both the API documentation and the end-user documentation) for Erebot is released under the CC BY-NC-SA license (see http://creativecommons.org/licenses/by-sa/3.0/).

The code itself is released under the GNU General Public License.

```
Copyright © 2010 François Poirotte

Erebot is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

Erebot is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with Erebot.  If not, see <http://www.gnu.org/licenses/>.
```

8.2 PEAR

php/PEAR/Exception.php is bundled with every Erebot PHAR archive as it is sometimes needed by other bundled packages (it may be removed later).

PEAR is released under the “2-clause BSD license”.

```
Copyright (c) 1997-2009,
 Stig Bakken <ssb@php.net>,
 Gregory Beaver <cellog@php.net>,
 Helgi Þormar Þorbjörnsson <helgi@php.net>,
 Tomas V.V.Cox <cox@idecnet.com>,
 Martin Jansen <mj@php.net>.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

8.3 File_Gettext

This package is used by Erebot to parse MO files (translation files). Only the files marked with the “php” or “data” role are bundled with Erebot’s phar.

File_Gettext is released under the “2-clause BSD license” since November, 8th 2005.

```
Copyright (c) 2004-2005, Michael Wallner <mike@iworks.at>.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
```

```
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

8.4 Console_CommandLine

This package is used to parse options given to Erebot's CLI. Only files with the "php" or "data" role are bundled with Erebot's phar archive.

Console_CommandLine is released under the MIT (Expat) license.

```
Copyright (c) 2007 David JEAN LOUIS, Richard Quadling

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

8.5 Symfony's Dependency Injection Container

We use our own special fork of Symfony's Depedency Injection Container (DIC) for Erebot to inject runtime dependencies. The fork uses the same license as the original project.

Symfony's dependency injection container is released under the MIT (Expat) license.

```
Copyright (c) 2008-2009 Fabien Potencier

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

8.6 Symfony's YAML component

Symfony's YAML component is released under the MIT (Expat) license.

```
Copyright (c) 2008-2009 Fabien Potencier

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

8.7 Composer

Used for runtime dependency checks when Erebot is used as a PHAR archive. Composer is released under the MIT (Expat) license.

```
Copyright (c) 2011 Nils Adermann, Jordi Boggiano

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

CHAPTER 9

Installation

This pages contains instructions on how to install this module on your machine. There are several ways to achieve that. Each method is described below.

Warning: You cannot mix the different methods. Especially, **you must use the same method to install this module as the one you selected for Erebot itself**.

Contents

Note: We recommend that you install this module using either its *PHAR package* or through *composer*. Installation from sources is reserved for advanced installations (eg. Erebot developers).

9.1 Installation using PHAR packages

Installing Erebot from a PHAR package is very easy. However, please note that Erebot must have been installed as a PHAR package for this method to work properly.

9.1.1 Preparations

If you haven't done so already, create a directory in Erebot's folder named `modules`.

Hence, your tree should look like this:

- **Erebot/**
  - Erebot-X.Y.Z.phar
  - modules/

Also, make sure your installation fulfills all of the *Prerequisites* for this module.

9.1.2 Downloading the package

First, select the version you want to install. Available versions are listed on *Erebot's package repository <https://packages.erebot.net/>*.

The PHAR package for a certain version can be downloaded by using a URL such as `https://packages.erebot.net/get/Erebot-version.phar` (replace *version* with the actual version you selected).

As a special shortcut, the following link always points to the latest snapshot of Erebot: https://packages.erebot.net/get/Erebot-dev-master.phar.

Warning: Using the latest snapshot available means that you may benefit from very recent developments, but it also means that the code may be in an unstable state. Use at your own risk.

The PHAR package must be downloaded to your installation's `modules/` directory.

9.1.3 Downloading the package's signature

All the packages delivered by Erebot's developers are cryptographically signed using the "OpenSSL" algorithm in PHP's Phar extension. This signature is used to detect corrupted packages and packages that have been tampered with.

You must retrieve the signature corresponding to the version of the PHAR package you downloaded and put it alongside the package. The signature can be downloaded by appending `.pubkey` at the end of the link to the package itself. Therefore, the signature for the latest version can be downloaded from https://packages.erebot.net/get/Erebot-dev-master.phar.pubkey.

Note: PHP automatically checks the integrity of signed PHAR packages when they are loaded. Neither the name of the PHAR package nor the name of the signature file should be altered, as the integrity check would then fail.

Warning: Although PHP automatically checks the integrity of cryptographically signed phar archives when they are loaded using the signature file, you may also check an archive manually by using the **`phar`** command provided with the phar extension.
For example, the following session shows a passing result.

```
$ phar info -f Erebot-dev-master.phar
# Alias:              Erebot
# Hash-type:          OpenSSL
# ... (other fields removed for clarity) ...
```

Note how the "Hash-type" field indicates that the "OpenSSL" algorithm has been used to sign the archive. **Any other value should be considered as if the check had failed**, unless the package was downloaded from Erebot's website over a secure (SSL/TLS) connection.
On the other hand, the following example shows a session where the verification failed.

```
$ phar info -f Erebot-dev-master.phar
# Exception while opening phar `Erebot-dev-master.phar':
# phar ``Erebot-dev-master.phar'' openssl signature could not be verified: openssl public
```

9.1.4 Conclusion

Once the PHAR package and its signature have been downloaded, your installation should look somewhat like that:

```
Erebot/
    Erebot-X.Y.Z.phar
    modules/
        Erebot-0.6.0.phar
        Erebot-0.6.0.phar.pubkey
```

That's all folks! You may now add *configuration options* for this module in Erebot's configuration file.

9.2 Installation through Composer

Installation through *Composer <http://getcomposer.org/>* is very easy. However, please note that Erebot itself must have been installed using Composer for this method to work properly.

To install the new module:

- Go to the directory where you installed Erebot.
- Add the module to your installation's dependencies with:

  ```
  $ # Replace 0.6.0 with whatever version you want to install.
  $ php composer.phar require --no-update erebot/erebot=0.6.0
  $ php composer.phar update --no-dev
  ```

- You may now add *configuration options* for this module in Erebot's configuration file.

9.3 Installation from source

Please note that Erebot itself must have been installed from source for this method to work.

First, make sure the git client is installed on your machine.

Under Linux, **from a root shell**, run the command that most closely matches the tools provided for your distribution:

```
# For apt-based distributions such as Debian or Ubuntu
$ apt-get install git

# For yum-based distributions such as Fedora, RHEL (RedHat) or CentOS
$ yum install git

# For urpmi-based distributions such as SLES (SuSE) or MES (Mandriva)
$ urpmi git
```

Note: Windows users may be interested in installing *Git for Windows <http://msysgit.github.io/>* to get an equivalent git client. Also, make sure that the path to `git.exe` is present on your account's `PATH`. Otherwise, you'll have to replace **`git`** with the full path to `git.exe` on every invocation. Eg. :

```
"C:\Program Files\Git\bin\git.exe" clone ...
```

Now, clone the module's repository:

```
$ cd /path/to/Erebot/vendor/
$ mkdir -p erebot
$ git clone git://github.com/Erebot/Erebot.git erebot/erebot
```

Last but not least, install the rest of this module's *Prerequisites* and then run:

```
$ cd /path/to/Erebot/vendor/erebot/erebot
$ /path/to/phing
```

You may now add *configuration options* for this module in Erebot's configuration file.

Badges:

D

E

P

R