Erebot

Release latest-0.6.0-40-gfa8e404

May 06, 2017

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 and its sub-components on the following websites:

- The official website
- Source repository
- Continuous Integration
- API documentation

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.

For now, Erebot is known to work on all PHP versions >= 5.3.3. Also, Erebot should run correctly on both Windows and most Linux distributions.

Table of Contents

Dependencies for regular users

The following table lists the PHP extensions that need to be available for the bot to work correctly.

Most of these extensions are actually part of PHP's core extensions and are thus usually enabled by default. Nonetheless, in case you compiled PHP yourself, you may need to recompile it to include the necessary extensions.

Table 1.1: Required PECL extensions for regular users

Dependency	Description
ctype	The `ctype` extension provides functions to test various properties of an input string.
DOM	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	Provides several helper classes to ease work on I18N (internationalization) in PHP applications.
libxml	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.
PCRE	Provides Perl-Compatible Regular Expressions for PHP.
Phar[4]	This extension is used to access a PHP Archive (phar) files.
POSIX[5]	Provides access to several functions only featured by POSIX-compliant operating systems.
Reflection	This extension makes it possible for some PHP code to inspect its own structure.
SimpleXML	Wrapper around libxml2 designed to make working with XML documents easier.
sockets	This extensions provides networking means for PHP applications.
SPL	The Standard PHP Library provides several functions and classes meant to deal with common usage patterns, improving code reuse.
One of mbstring, iconv, recode or XML	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 and are thus recommended over the other extensions.

Additional dependencies for Erebot developers

These dependencies are only necessary if you want to participate into the bot's development. For regular usage, refer to the list of *Dependencies for regular users*.

Erebot developers should install both the dependencies from the regular set plus the ones listed below in order to get a working setup.

For Linux, those dependencies can usually be installed by issuing one of the following commands **as a privileged user**:

```
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 recent versions of Fedora, RedHat and CentOS.
root@localhost:~# dnf install <package>
```

[1] Needed if you want to connect to IRC servers using a secure (encrypted) connection. Also required when running Erebot from a PHAR archive to check the archive's integrity.
[2] Required for daemonization and to change user/group information upon startup. (not available on Windows)
[4] Required to run Erebot from a `.phar` archive.
[5] Required to change user/group information upon startup. (not available on Windows)

```
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>
```

Please refer to your distribution's documentation for more information.

For Windows, each dependency must be downloaded and installed separately as there is no central package manager like on Linux.

System dependencies

The following table lists the necessary system dependencies required for Erebot's development. For apt-based systems like Debian/Ubuntu, an installation link is also provided. Instructions for Windows users are also provided.

Table 1.2: System dependencies for developers

Dependency	APT link	Description
doxygen	Install this package	`doxygen` is needed if you plan to generate the documentation from Erebot's source files. We recommend version 1.8.0 or later. Windows users may download a pre-built binary release from Doxygen's download page.
gettext	Install this package	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. Installing gettext for Windows is a bit more tedious. 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 (eg. no 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.
xmlstarlet	Install this package	`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. Windows users may download a pre-build binary release from the project's download page.

PHP extensions

The following table lists additional PHP extension that need to be installed by Erebot developers.

Table 1.3: Required PECL extensions for Erebot developers

Dependency	Description
Phar[3]	This extension is used to create a PHP Archive (phar) containing the bot's code, providing users with an easy way to install Erebot. This extension is part of PHP core on Windows and so, Windows users don't need to do anything specific to benefit from it.
xdebug[6]	Debugging execution of PHP code is made possible by this extension. It is also used to retrieve code coverage information while testing the code. Pre-built binary releases for Windows can be downloaded from Xdebug's download page. Make sure to download the build matching your PHP installation (same VC version, same thread-safe support & same architecture).
XSL	The XSL extension implements the XSL standard, performing XSLT transformations using the libxslt library. This extension is bundled with PHP on Windows and so, Windows users only need to activate it through the `php.ini` configuration file.

[3] Required to package Erebot as a `.phar` archive.
[6] Only required to run the test suite.

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**.

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 the following modules:

- Erebot_Module_AutoConnect
- Erebot_Module_IrcConnector
- Erebot_Module_PingReply

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

- 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.

- Download the PHAR archive for Erebot itself. You can grab the latest version from our *release page <github.com/Erebot/Erebot/releases/latest/>*. You MUST also download the public signature for the archive. The signature is available for download alongside the PHAR archive.
- Create a directory named `modules` in the same folder as the PHAR.
- Download the PHAR archive and its signature for each of the following modules:
  - Erebot_Module_AutoConnect
  - Erebot_Module_IrcConnector
  - Erebot_Module_PingReply

  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.

  Make sure you also read each module's documentation to look for any additional prerequisites.
- (Optional) Download the PHAR archives & signature of any additional module you would like to use.

Your tree should now look like this:

- **Erebot/**
  - `Erebot-`*`X.Y.Z`*`.phar`
  - `Erebot-`*`X.Y.Z`*`.phar.pubkey`
  - **modules/**
    - `Erebot_Module_AutoConnect-`*`X.Y.Z`*`.phar`
    - `Erebot_Module_AutoConnect-`*`X.Y.Z`*`.phar.pubkey`
    - `Erebot_Module_IrcConnector-`*`X.Y.Z`*`.phar`
    - `Erebot_Module_IrcConnector-`*`X.Y.Z`*`.phar.pubkey`
    - `Erebot_Module_PingReply-`*`X.Y.Z`*`.phar`
    - `Erebot_Module_PingReply-`*`X.Y.Z`*`.phar.pubkey`
    - *eventually, additional PHAR archives with their signature*

Once the PHAR archives have been retrieved, you may wish to change file permissions on `Erebot-`*`X.Y.Z`*`.phar`:

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

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

```
$ ./Erebot-*.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.

Installation using Composer

First, make sure Composer is installed. If not, follow the *installation instructions <https://getcomposer.org/download/>* on their website.

- Create a new folder named `Erebot` and go into that folder:

  ```
  me@localhost:~/$ mkdir Erebot
  me@localhost:~/$ cd Erebot
  ```

- Use Composer to install the bot's code:

  ```
  me@localhost:~/Erebot/$ php /path/to/composer.phar require --update-no-dev erebot/
  ↪erebot erebot/ircconnector-module erebot/pingreply-module erebot/autoconnect-
  ↪module
  ```

  You may pass additional module names if you want to use other modules.

- Next, if you're an Erebot developer, install development dependencies as well:

  ```
  me@localhost:~/Erebot/$ php /path/to/composer.phar update
  ```

That's it! The bot is now installed. Be sure to read the section on *final steps* for a summary of what to do next.

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 using Composer.
# Must be run from the folder in which Erebot was installed.
$ php ./vendor/bin/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

Configuration

Erebot's configuration is stored in an XML file. This file is usually called `Erebot.xml` but can be renamed (see *`Erebot -c`*).

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

  - *<networks>*
    - *<network>*
      - *<servers>*
      - *<server>*
      - *<channels>*
      - *<channel>*

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.

<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.

<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.

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

<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.

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.

<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.com"/>
```

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

Usage

Options

-c <FILE>, **--config** <FILE>
Use the given configuration <FILE> instead of the default `Erebot.xml`. The given path is interpreted relative to the current directory.

-d, --daemon
Run the bot in the background.

Note: Both the POSIX and pcntl PHP extensions must be enabled for this option to work.

-n, --no-daemon
Run the bot in the foreground. This is the default unless *`Erebot -d`* is used.

-p <FILE>, **--pidfile** <FILE>
Path to the file where the bot's PID (Process IDentifier) will be written, relative to the current directory.

-g <GROUP/GID>, **--group** <GROUP/GID>
Switch to this group identity during startup. The group may be expressed as either a group name (eg. `root`) or as a numeric GID (Group IDentifier) (eg. `0`).

-u <USER/UID>, **--user** <USER/UID>
Switch to this user identity during startup. The user may be expressed as either a user name (eg. `root`) or as a numeric UID (User IDentifier) (eg. `0`).

-h, --help
Display the bot's help and exit.

-v, --version
Display the bot's version and exit.

Environment variables

Several environment variables related to language settings control the way the bot produces its output. The first variable to appear in the environment takes precedence over the others.

LANGUAGE
Defines the system's supported languages, as a colon-separated list of country names with optional regions. Eg.

```
LANGUAGE=en_US:en
```

LC_ALL
Defines supported languages for various types of formatting operations, like message formatting, date/time formatting and so on. Setting this variable is equivalent to setting each of the other `LC_*` variables individually to the same value.

LC_MESSAGES
Defines supported languages when outputting textual messages.

LC_MONETARY
Defines supported languages when outputting monetary values.

LC_TIME
Defines supported languages when outputting dates/times.

LC_NUMERIC
Defines supported languages when outputting other numeric values (eg. floating-point values).

LANG
Defines the system's supported languages and encodings, as a colon-separated list of country names with their optional region and their encoding. Eg.

```
LANG=en_US.utf8
```

CHAPTER 5

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*

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
- TriggerRegistry
- TV

- Uno
- WatchList
- WebGetter
- Werewolves
- Wordlists

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

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 6

Tutorials

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

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

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 6.1: 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:

```
# m h  dom mon dow   command
  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.

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).

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 /home/clicky/Documents/Erebot/src/Prompt.php on line 44
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/
↪DependencyInjection/sfServiceContainer.php:0
PHP   4. sfServiceContainerBuilder->getService() /var/local/buildbot/pear/php/
↪SymfonyComponents/DependencyInjection/sfServiceContainer.php:276
PHP   5. sfServiceContainerBuilder->createService() /var/local/buildbot/pear/php/
↪SymfonyComponents/DependencyInjection/sfServiceContainerBuilder.php:86
PHP   6. ReflectionClass->newInstanceArgs() /var/local/buildbot/pear/php/
↪SymfonyComponents/DependencyInjection/sfServiceContainerBuilder.php:248
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' for '/tmp/Erebot.sock'' in /home/clicky/Documents/Erebot/src/Prompt.
↪php:56
Stack trace:
#0 [internal function]: \Erebot\Prompt->__construct(Object(Erebot), '/tmp/Erebot.soc..
↪.', 'nogroup', 384)
#1 /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjection/
↪sfServiceContainerBuilder.php(248): ReflectionClass->newInstanceArgs(Array)
#2 /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjection/
↪sfServiceContainerBuilder.php(86): sfServiceContainerBuilder->
↪createService(Object(sfServiceDefinition))
#3 /var/local/buildbot/pear/php/SymfonyComponents/DependencyInjection/
↪sfServiceContainer.php(276): sfServiceContainerBuilder->getService('prompt')
#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 7

Contribute

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

There are several ways by which you may contribute to the project.

Try it!

The more people use it, the better, because it means bugs and regressions can be detected more quickly.

Report Bugs / Suggest Features

If you use Erebot and find issues with it, please let us know on GitHub. Try to provide as much detail as possible on how to reproduce your issue. As a rule of thumb, the easier it is to reproduce an issue, the quicker it gets fixed.

Improve the Documentation

The documentation for the project is stored alongside the code, in the `docs/src/` folder. We use the Sphinx documentation builder and the Read The Docs platform to produce the documentation in a variety of output formats.

We try to document Erebot as much as we can, but acting both as developers and documentation writers, we tend to be biased as to what needs documentation.

So if you feel like some parts could be clearer, send us a pull request with your modifications and we'll try to review them as soon as possible. Any help to improve the documentation will be greatly appreciated!

New modules

If you wish to contribute new modules, you should probably start by reading our development guide on *Writing a new module*. 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*.

Pull requests

If you plan on sending us pull requests, please read our documentation on Erebot's *Coding standard* first. Your patch will have a greater chance of being merged if it already 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.
- Hack away 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, or if it needs some tweaking before being merged.

Translations

The project uses the Transifex service to manage translations. All submissions or patches to translations must 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 eventually be merged in the project alongside other translations.

Join the community

If you would like to chat with the developers and other users of Erebot, feel free to join our IRC channel. This channel can also be used to discuss issues, new ideas / feature requests and to follow the bot's development.

CHAPTER 8

Developer Zone

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

Coding standard

This page is only of interest for Erebot developers.

Table of Contents

Writing code

Erebot uses the PSR-2 coding style for PHP code.

Compliance with this standard can be tested by running the following command from the top directory:

```
phing qa_codesniffer
```

Writing documentation

For the code, we rely on Doxygen commands to automatically extract the API documentation. We use the \ prefix for such commands as it is also recognized by other tools. Also, we use the \copydoc command massively to avoid repeating ourselves while documenting the code.

The rest of the documentation (what you are currently reading) is managed using the Sphinx documentation generator.

All the documentation is stored in the same Git repository as the code to help keep both the code and documentation in sync.

Both documentation can be built by running this command from the top directory:

```
phing doc
```

Writing tests

We use the PHPUnit testing framework to write unit and functionnal tests for the bot.

The tests are stored in the same Git repository as the code to help keep both the code and its tests in sync.

The test suite can be run by using the following command from the top directory:

```
phing tests
```

Other tools

We also use other tools to measure various metrics of the code, like code complexity, code repetitions, and so on. The full QA (Quality Assurance) test suite can be run with:

```
phing qa
```

Internationalization

Table of Contents

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
<?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.

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.

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.

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.

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.

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.

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

  - `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).
  - `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.

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

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

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

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);

    // 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; ">'.
                    '<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
<?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!).

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

Quick start

To quickly create the file structure for a new module, you can start by cloning/forking the git repository for the Module_Skeleton virtual module. This will provide you with the basic structure and necessary files.

Please note that the Module_Skeleton module uses Composer to handle dependencies, and so, it is worth reading *Composer's documentation <https://getcomposer.org/doc/>* if you are new to PHP packages and dependency management.

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).

Providing help

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\CallableInterface interface and referring to your method.

This can be done using the following snippet:

```
// We register our help method (here, the getHelp() method
// from the current object) by wrapping a callback-compatible
// value referring to it a special wrapper object.
$this->registerHelpMethod(\Erebot\CallableWrapper::wrap(array($this, 'getHelp')));
```

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

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 (eg. ErebotModulePingReply). 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 9

Licenses

Table of Contents

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
```

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".

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.

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.

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.

Symfony's YAML component

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

Composer

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

```
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 10

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).

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.

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.

Downloading the package

First, select the version you want to install. Available versions are listed on Erebot's package repository.

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.

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 key could not be read
```

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-latest-0.6.0-40-gfa8e404.phar
        Erebot-latest-0.6.0-40-gfa8e404.phar.pubkey
```

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

Installation through Composer

Installation through Composer 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 this module to your installation's dependencies with:

  ```
  $ # Replace latest with whatever version you want to install.
  $ php composer.phar install erebot/erebot=latest
  ```

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

Installation from source

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

Warning: This method exists only for the sake of running Erebot on the now deprecated PHP 5.2.x. Also, please note that depending on your environment, other actions than the ones described here may be required to make this module work properly.

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 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`** by 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:

Index