Flycheck

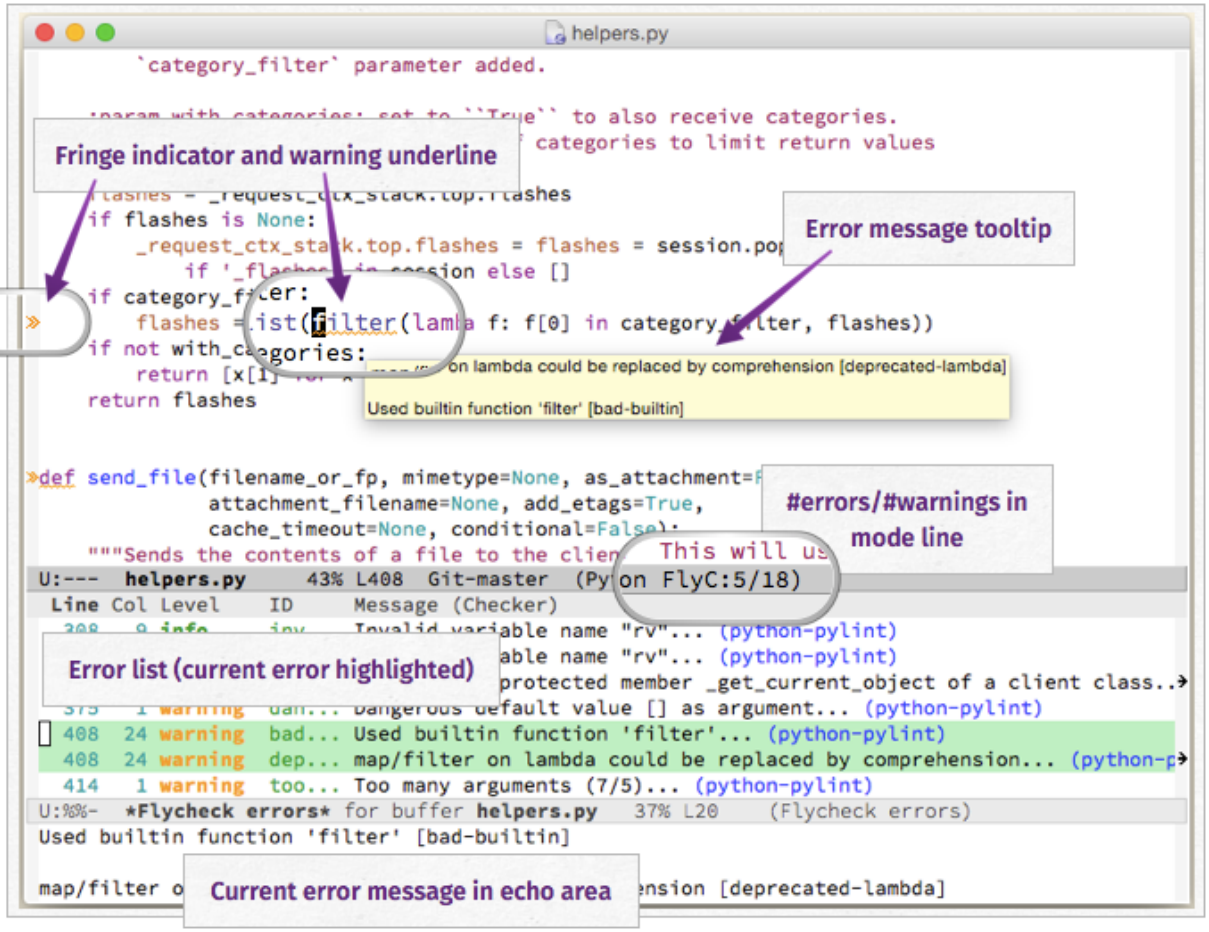
Release 32-cvs

Nov 05, 2020

Contents

Flycheck is a modern on-the-fly syntax checking extension for GNU Emacs, intended as replacement for the older Flymake extension which is part of GNU Emacs. For a detailed comparison to Flymake see *Flycheck versus Flymake*.

It uses various syntax checking and linting tools to *automatically check the contents of buffers* while you type, and reports warnings and errors directly in the buffer, or in an optional *error list*:



Out of the box Flycheck supports over *40 different programming languages* with more than 80 different syntax checking tools, and comes with a *simple interface* to define new syntax checkers.

Many *3rd party extensions* provide new syntax checkers and other features like alternative error displays or mode line indicators.

CHAPTER 1

Try out

Flycheck needs GNU Emacs 24.3+, and works best on Unix systems. **Windows users**, please be aware that Flycheck does not support Windows officially, although it should mostly work fine on Windows. See *Windows support* and watch out for known Windows issues!

To try Flycheck in your Emacs session install some *syntax checker tools* and type the following in your `*scratch*` buffer and run `M-x eval-buffer`:

```
(require 'package)
(add-to-list 'package-archives
             '("MELPA Stable" . "http://stable.melpa.org/packages/") t)
(package-initialize)
(package-refresh-contents)

(package-install 'flycheck)

(global-flycheck-mode)
```

On MacOS also add the following to *fix your $PATH environment variable*:

```
(package-install 'exec-path-from-shell)
(exec-path-from-shell-initialize)
```

For a permanent installation of Flycheck follow the *Installation* instructions. For a gentle introduction into Flycheck features go through *Quickstart* guide.

Important: If Flycheck fails to run properly or gives you any error messages please take a look at the *troubleshooting section* which covers some common setup issues and helps you debug and fix problems with Flycheck.

CHAPTER 2

The User Guide

The User Guide provides installation and usage help for Flycheck. It starts with installation instructions and a quick start tutorial and then focuses on an in-depth references of all parts of Flycheck.

2.1 Installation

This document gives you detailed instructions and information about installing Flycheck.

2.1.1 Prerequisites

Flycheck needs GNU Emacs 24.3+ and works best on Unix-like systems like Linux or macOS. It does not support older releases of GNU Emacs or other flavours of Emacs (e.g. XEmacs, Aquamacs, etc.).

Windows support

Flycheck does not explicitly support Windows, but tries to maintain Windows compatibility and should generally work fine on Windows, too. However, we can neither answer questions about Windows nor fix bugs that only occur on Windows without the help of active Windows users. Please watch out for known Windows issues.

Syntax checking tools

Flycheck does not check buffers itself but relies on *external* programs to check buffers. These programs must be installed separately. Please take a look at the *list of supported languages* to find out what tools are required for a particular language.

Many of these programs are available in the package repositories of Linux distributions or in Homebrew for macOS. Others can be installed with standard package managers such as Rubygems, NPM, Cabal, etc.

Important: For a GUI Emacs on MacOS we recommend to install and configure exec-path-from-shell to make Emacs use the proper `$PATH` and avoid a *common setup issue on MacOS*.

2.1.2 Package installation

We recommend to install Flycheck with Emacs' built-in package manager. Flycheck is available in the popular MELPA archive which serves up to date snapshots of Flycheck's development state. We recommend to read through the *changelog* before every upgrade to check for any breaking changes that might affect you.

Note: The sibling repository MELPA Stable provides packages for Flycheck releases. If you prefer to follow the most recent changes use MELPA instead, but be aware that, while we try to be careful about the stability of the development snapshots, we may make breaking changes anytime without prior announcement.

Unfortunately the MELPA repositories are not available in Emacs by default. You must explicitly add them to `package-archives` with the following code in your *init file*:

```
(require 'package)

(add-to-list 'package-archives
             '("MELPA Stable" . "https://stable.melpa.org/packages/") t)
(package-initialize)
```

This adds MELPA Stable; for MELPA replace `https://stable.melpa.org` with `https://melpa.org` and change the name accordingly. If you do not know where your init file is inspect the value of `user-init-file` with `C-h v user-init-file`.

Once the repository is set up you can install Flycheck from Emacs' package menu at `M-x list-packages`, or directly with `M-x package-install RET flycheck`.

use-package

You may want to take a look at use-package which provides simple syntax to declare and configure packages in your init file. Specifically it allows to automatically install missing packages from package archive when Emacs starts.

Add the following form to your init file to setup Flycheck with use-package:

```
(use-package flycheck
  :ensure t
  :init (global-flycheck-mode))
```

Then press `C-M-x` with point somewhere in this form to install and enable Flycheck for the current Emacs session.

Distribution packages

Alternatively some distributions provide binary packages of Flycheck. We officially support the following distributions:

- Debian 9 and newer: `apt-get install elpa-flycheck flycheck-doc` (the latter for our manual). The Debian Emacs addon team provides these packages.

2.1.3 Legacy installation methods

Some users prefer to install Flycheck with legacy methods such as el-get, Git submodules, etc that were common before Emacs included a package manager. There are also many 3rd party packages provided by various package managers. We do neither support nor endorse any of these:

> **Warning:** If you install Flycheck in any way other than *our official packages* you do so **at your own risk**.

Please beware of breakage, and understand that while we do not actively work against alternative installation methods we will not make compromises to support alternative installation methods. We will close issues reported for alternative installation if we fail to reproduce them with a proper installation of Flycheck.

2.2 Quickstart

This page gives a quick introduction into Flycheck and an overview of its most important features. Before you start here please make sure that Flycheck is *installed*.

2.2.1 Enable Flycheck

Now add the following code to your *init file* to permanently enable syntax checking with Flycheck:

```
(add-hook 'after-init-hook #'global-flycheck-mode)
```

2.2.2 Install syntax checker programs

Now you need to install syntax checking programs for the languages you'd like to use Flycheck with. The *list of supported languages* tells you which languages Flycheck supports and what programs it uses.

For instance, you can install Pylint for Python and ESLint for Javascript:

```
$ pip install pylint
$ npm install eslint
```

2.2.3 Check syntax in a buffer

Now you are ready to use Flycheck in a Python or Javascript buffer. Visit a Python or Javascript file and check whether your Flycheck setup is complete with `C-c ! v`.

If everything is green, Flycheck will now start to check the buffer on the fly while you are editing. Whenever you make a mistake that eslint or Pylint can catch, Flycheck will highlight the corresponding place in the buffer with an error underline whose color reflects the severity of the issue. Additionally, Flycheck will put a symbol into the fringe for affected lines and show the total number of errors and warnings in the buffer in the mode line.

2.2.4 Navigate and list errors

With `C-c ! n` and `C-c ! p` you can now jump back and forth between erroneous places. If you keep on such a place for a little while Flycheck will show the corresponding error message in the each area. Likewise, if you hover such a place with the mouse cursor Flycheck will show the error message in a tooltip.

Press `C-c ! l` to pop up a list of all errors in the current buffer. This list automatically updates itself when you fix errors or introduce new ones, and follows the currently selected buffer. If the error list is selected you can type `n` and `p` to move up and down between errors and jump to their corresponding location in the buffer.

2.2.5 More features

All Flycheck commands are available in the Emacs Menu at *Tools -→ Syntax checking*:

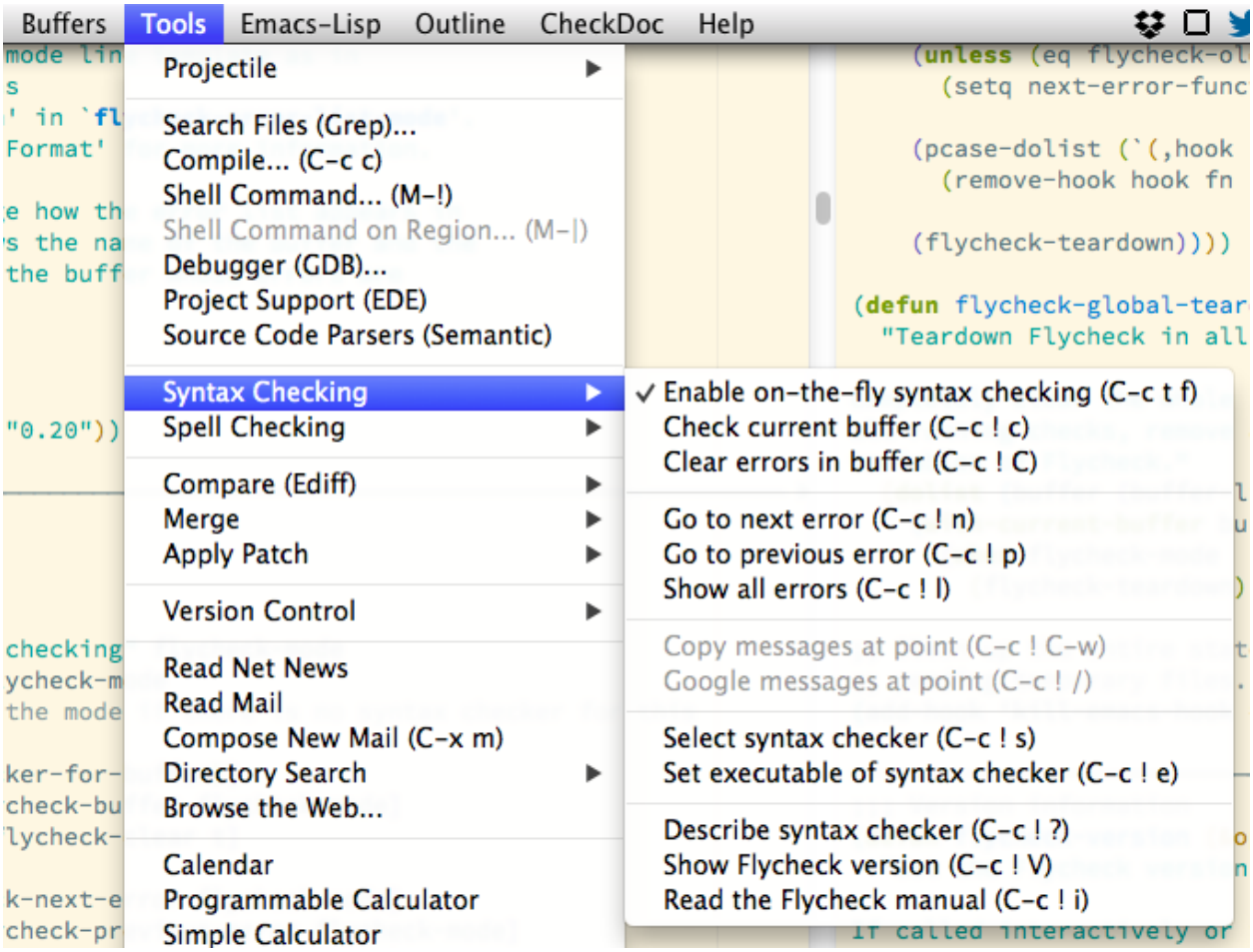


Fig. 1: The menu of Flycheck, showing all available Flycheck commands

The same menu also pops up when you click on the mode line lighter:

2.3 Troubleshooting

If syntax checking does not work as expected there are a number of steps that you can follow to isolate and maybe fix the problem.

2.3.1 Common issues

First check whether your issue is one of the common setup issues and problems.

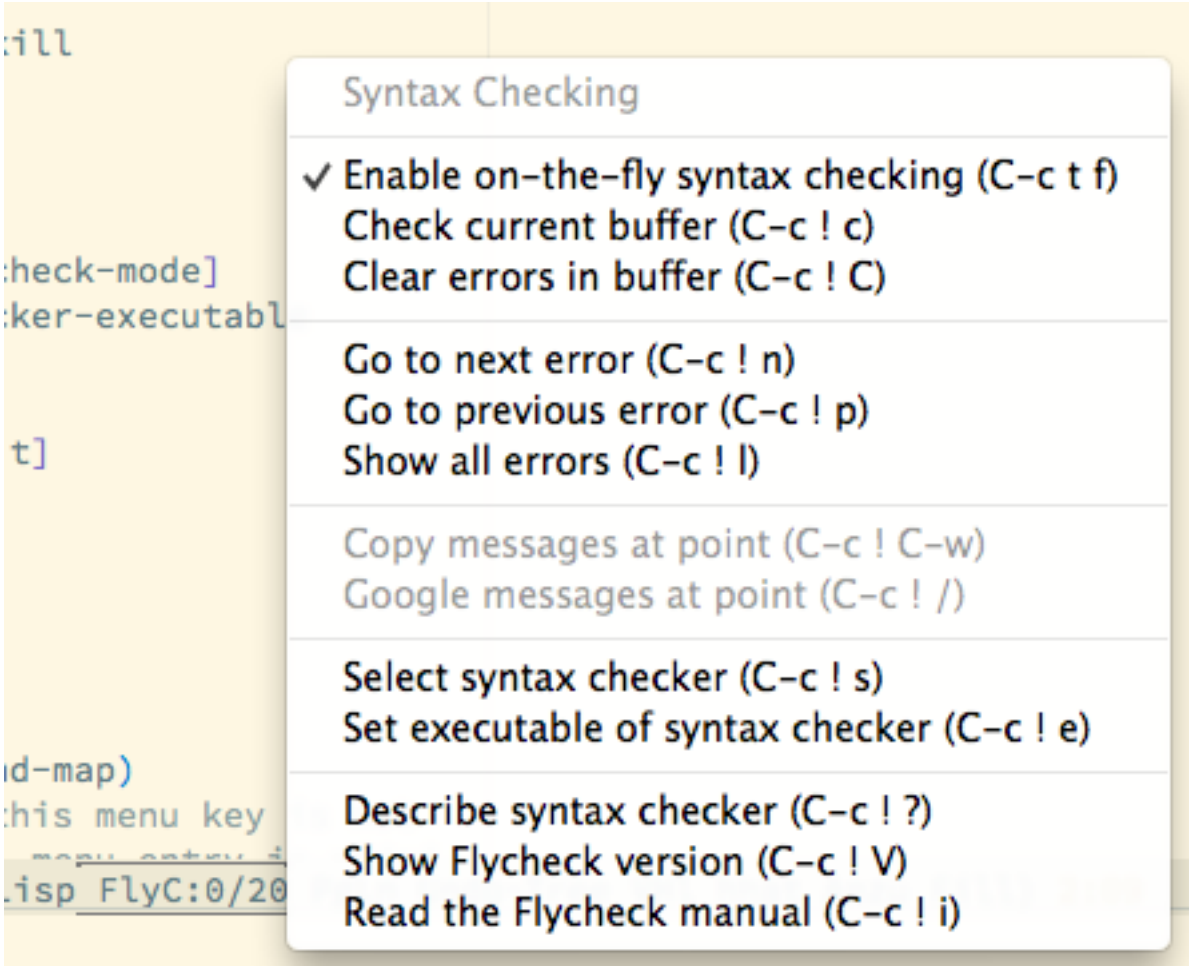


Fig. 2: The mode line menu of Flycheck

Flycheck can't find any programs in GUI Emacs on MacOS

Try to install and configure exec-path-from-shell to make a GUI Emacs inherit the `$PATH` environment variable from your shell configuration.

The issue is that due to the special way MacOS starts GUI programs a GUI Emacs does not inherit the environment variables from the shell configuration so Emacs will lack some important entries in `$PATH`, most notably `/usr/local/bin/` where Homebrew, NPM and many other package managers put binaries in.

The exec-path-from-shell works around this issue by extracting environment variables from a shell session and inject them into the environment of the running Emacs instance.

Flycheck warns about "non-zero exit code, but no errors"

Make sure that you have the latest version of the syntax checker installed, particularly if the message started appearing after you updated Flycheck.

Newer releases of Flycheck may require newer versions of syntax checking tools. For instance Flycheck might now pass a command line flag that older versions do not understand, or attempt to parse an updated output format. In these cases the syntax checker will show an error message about an unknown flag, or emit output that Flycheck does not understand, which prompts Flycheck to warn that even though the syntax checker appeared to not have successfully checked the buffer content there are no errors to be found.

If you *are* using the latest version then this message most likely indicates a flaw in the syntax checker definition. In this case please *report a bug* to us so that we can fix the issue. Please don't forget to say that you are using the latest version!

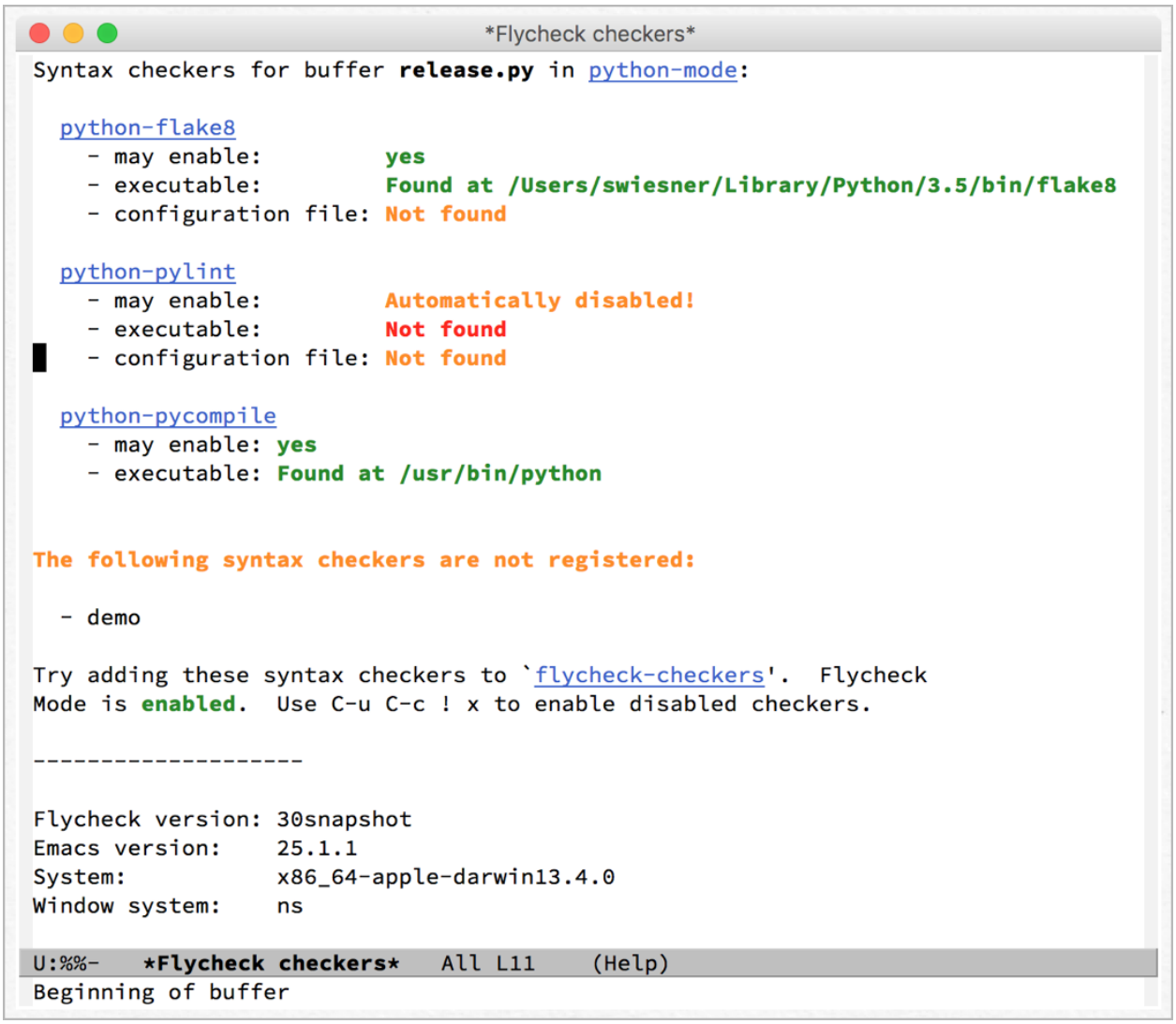
2.3.2 Verify your setup

If your issue is none of the aforementioned *common issues* the first step is to let Flycheck check your setup:

C-c ! v
M-x flycheck-verify-setup
Show a *verification buffer* with information about your *Flycheck Mode* setup for the current buffer.

The buffer contains all syntax checkers available for the current buffer and tells you whether Flycheck would use each one and what reasons would prevent Flycheck from using a checker. It also includes information about your Flycheck and Emacs version and your operating system.

The following image shows a *verification buffer*:



The buffer shows all syntax checkers for the current buffer. Note that you can click on the syntax checker names to show the docstring for a syntax checker.

- *Green* items indicate *good* configuration. In the screenshot both `python-flake8` and `python-pycompile` exist.
- *Orange* items indicate a *potential* misconfiguration. The screenshot shows that no configuration file was found for `python-flake8` which is perfectly fine if there's no flake8 configuration file in the project, but not so good if you'd like Flycheck to use a configuration file for flake8. The section *Configuration files* has more information about configuration files.

  Likewise the buffer warns you that a `demo` syntax checker (which is not part of Flycheck of course) isn't registered in `flycheck-checkers`. If you'd like Flycheck to automatically use this syntax checker you should fix this issue by adding it to `flycheck-checkers` but otherwise it's safe to ignore this warning.
- *Red* items indicate *bad* configuration. `python-pylint` wasn't found in the screenshot, so you'll not be able to use pylint in the current buffer.

2.3.3 Debug syntax checkers

If a syntax checker fails although it successfully verified you need to take a closer look. Flycheck provides you with a command that lets you run a single syntax checker just the way Flycheck would run it:

C-c ! C-c
M-x flycheck-compile
Prompt for a syntax checker and run in as a shell command, showing the whole output in a separate buffer.

Important: The current implementation this command suffers from a couple of issues, so we'd like to have a replacement in GH-854 and we could use your help! If you'd like to help out with this task please join the discussion in that issue.

The output of this command can provide you helpful clues about what's going on. It also helps to compare the output of the command in Emacs with what happens if you run the same command in a terminal.

2.3.4 If all else fails...

...please do *ask for help*. We have many different channels, from Twitter to a chat room to StackOverflow, whatever suits you best, and we try to help you as fast and as well as possible.

2.4 Check buffers

Flycheck provides two Emacs minor modes for automatic syntax checking: *Flycheck Mode* to enable syntax checking in the current buffer, and *Global Flycheck Mode* to enable syntax checking in all buffers whenever possible.

Minor Mode Flycheck Mode
Enable *automatic syntax checking* in the current buffer.

Minor Mode Global Flycheck Mode
Enable *Flycheck Mode* in all buffers where syntax checking is possible.

Note: This mode does not enable *Flycheck Mode* in remote files (via TRAMP) and encrypted files. Checking remote files may be very slow depending on the network connections, and checking encrypted files would leak confidential data to temporary files and subprocesses.

You can manually enable *Flycheck Mode* in these buffers nonetheless, but we do *not* recommend this for said reasons.

Add the following to your *init file* to enable syntax checking permanently:

```
(add-hook 'after-init-hook #'global-flycheck-mode)
```

You can exclude specific major modes from syntax checking with *flycheck-global-modes*:

defcustom flycheck-global-modes
Major modes for which *Global Flycheck Mode* turns on *Flycheck Mode*:

t (the default) Turn *Flycheck Mode* on for all major modes.

(*foo-mode* ...) Turn *Flycheck Mode* on for all major modes in this list, i.e. whenever the value of `major-mode` is contained in this list.

(not *foo-mode* ...) Turn *Flycheck Mode* on for all major nodes *not* in this list, i.e. whenever the value of `major-mode` is *not* contained in this list.

Note: *Global Flycheck Mode* never turns on *Flycheck Mode* in major modes whose `mode-class` property is `special`, regardless of the value of this option. Syntax checking simply makes no sense in special buffers which are typically intended for non-interactive display rather than editing.

See also:

Major Mode Conventions(elisp) Information about major modes, and modes marked as special.

2.4.1 Check automatically

By default *Flycheck Mode* automatically checks a buffer whenever

- it is enabled,
- the buffer is saved,
- a new line is inserted,
- or a short time after the last change was made in a buffer.

You can customise this behaviour with *flycheck-check-syntax-automatically*:

defcustom flycheck-check-syntax-automatically
A list of events which trigger a syntax check in the current buffer:

save Check the buffer immediately after it was saved.

new-line Check the buffer immediately after a new line was inserted.

idle-change Check the buffer a short time after the last change. The delay is customisable with *flycheck-idle-change-delay*:

defcustom flycheck-idle-change-delay
Seconds to wait after the last change to the buffer before starting a syntax check.

idle-buffer-switch Check the buffer a short time after switching to it from another buffer. The delay is customisable with *flycheck-idle-buffer-switch-delay*:

defcustom flycheck-idle-buffer-switch-delay
Seconds to wait after switching to a buffer before starting a syntax check.
If you switch to several buffers in rapid succession, the behavior depends on *flycheck-buffer-switch-check-intermediate-buffers*:

defcustom flycheck-buffer-switch-check-intermediate-buffers
If non-nil, then a buffer you switch to will have a syntax check run even if you switch to another buffer before it starts. If nil, then only the current buffer can have a syntax check run. Note that syntax checks can still be run in other buffers due to changes to their contents.

mode-enabled Check the buffer immediately after *Flycheck Mode* was enabled.

For instance with the following setting *Flycheck Mode* will only check the buffer when it was saved:

```
(setq flycheck-check-syntax-automatically '(mode-enabled save))
```

2.4.2 Check manually

You can also start a syntax check explicitly with *C-c ! c*:

C-c ! c
M-x flycheck-buffer
Check syntax in the current buffer.

2.5 Syntax checkers

Flycheck does not check buffers on its own. Instead it delegates this task to external *syntax checkers* which are external programs or services that receive the contents of the current buffer and return a list of errors in the buffer, together with metadata that tells Flycheck how to run the program, how to pass buffer contents to it, and how to extract errors.

See also:

Supported Languages A complete list of all syntax checkers included in Flycheck

Like everything else in Emacs syntax checkers have online documentation which you can access with *C-c ! ?*:

C-c ! ?
M-x flycheck-describe-checker
Prompt for the name of a syntax checker and pop up a Help buffer with its documentation.

The documentation includes the name of the program or service used, a list of major modes the checker supports and a list of all options for this syntax checker.

2.5.1 Select syntax checkers automatically

Normally Flycheck automatically selects the best syntax checkers for the current buffer from *flycheck-checkers* whenever it needs to check the buffer:

defcustom flycheck-checkers
A list of all syntax checkers available for syntax checking.

A syntax checker in this list is a *registered syntax checker*.

Flycheck picks the first syntax checker from this list which exists and supports the current major mode, and runs it over the current buffer. When the checker has finished, Flycheck looks for the next syntax checker to run, and if there is one, Flycheck runs the next syntax checker, and so on, until there is no more syntax checker for the current buffer. This process repeats whenever Flycheck needs to check the buffer according to *flycheck-check-syntax-automatically*.

Important: Under some circumstances—for instance if the syntax checker is not installed— Flycheck automatically *disables syntax checkers* in the current buffer and will thus not even consider them in any future checks in the current buffer.

In the *verification buffer* these syntax checkers are marked as "disabled" just as if you had disabled them manually with *C-c ! x*, and likewise you can re-enable automatically disabled syntax checkers with *C-u C-c ! x*.

For instance, the first syntax checker for Emacs Lisp is *emacs-lisp* which checks Emacs Lisp with Emacs' own byte compiler. This syntax checker asks for *emacs-lisp-checkdoc* to run next, which checks for stylistic issues in Emacs Lisp docstrings. Thus Flycheck will first run the byte compiler and then checkdoc in an Emacs Lisp buffer.

2.5.2 Select syntax checkers manually

Alternatively you can tell Flycheck explicitly which syntax checker to start with in the current buffer:

C-c ! s
M-x flycheck-select-checker
Prompt for a syntax checker and use this syntax checker as the first syntax checker for the current buffer.

Flycheck may still run further syntax checkers from *flycheck-checkers* if the selected syntax checker asks for it.

Flycheck will use the selected syntax checker as "entry point" for syntax checks in the current buffer, just as if it had selected this syntax checker automatically. It will automatically run further syntax checkers from *flycheck-checkers* if the selected syntax checker asks for it.

Under the hood *C-c ! s* sets *flycheck-checker*:

defvar flycheck-checker
The name of a syntax checker to use for the current buffer.

If `nil` (the default) let Flycheck *automatically select* the best syntax checker from *flycheck-checkers*.

If set to a syntax checker Flycheck will use this syntax checker as the first one in the current buffer, and run subsequent syntax checkers just as if it had selected this one automatically.

If the syntax checker in this variable does not work in the current buffer signal an error.

This variable is buffer-local.

We recommend to set *flycheck-checker* via directory local variables to enforce a specific syntax checker for a project. For instance, Flycheck usually prefers *javascript-eslint* for Javascript buffers, but if your project uses *javascript-jshint* instead you can tell Flycheck to use *javascript-jshint* for all Javascript buffers of your project with the following command in the top-level directory of your project: `M-x add-dir-local-variable RET js-mode RET flycheck-checker RET javascript-jshint`. A new buffer pops up that shows the newly created entry in the directory variables. Save this buffer and kill it. From now on Flycheck will check all Javascript files of this project with JSHint.

See also:

Locals(emacs) General information about local variables.

Directory Variables(emacs) Information about directory variables.

To go back to automatic selection either set *flycheck-checker* to `nil` or type *C-u C-c ! s*:

C-u C-c ! s
C-u M-x flycheck-select-checker
Remove any selected syntax checker and let Flycheck again *select a syntax checker automatically*.

2.5.3 Disable syntax checkers

Even if you *select a checker manually* Flycheck may still use a syntax checker that you'd not like to use. To completely opt out from a specific syntax checker disable it:

C-c ! x
M-x flycheck-disable-checker
Prompt for a syntax checker to disable in the current buffer.

For instance if you do not care for documentation conventions of Emacs Lisp you can opt out from *emacs-lisp-checkdoc* which checks your code against these conventions with `C-c ! x emacs-lisp-checkdoc`. After the next check all checkdoc warnings will be gone from the buffer.

Internally this command changes the buffer-local *flycheck-disabled-checkers*:

defcustom flycheck-disabled-checkers
A list of disabled syntax checkers. Flycheck will *never* use disabled syntax checkers to check a buffer.

This option is buffer-local. You can customise this variable with `M-x customize-variable RET flycheck-disabled-checkers` or set the default value in your *init file* to permanently disable specific syntax checkers. For instance:

```
(setq-default flycheck-disabled-checkers '(c/c++-clang))
```

will permanently disable *c/c++-clang* in all buffers.

You can also disable syntax checkers per project with directory local variables. For instance type `M-x add-dir-local-variable RET emacs-lisp-mode RET flycheck-disabled-checkers RET (emacs-lisp-checkdoc)` in your *user emacs directory* to disable *emacs-lisp-checkdoc* for all Emacs Lisp files in your personal configuration.

See also:

Locals(emacs) General information about local variables.

Directory Variables(emacs) Information about directory variables.

To enable a disabled checker again, remove it from *flycheck-disabled-checkers* or use *C-u C-c ! x*:

C-u C-c ! x
C-u M-x flycheck-disable-checker
Prompt for a disabled syntax checker to enable again in the current buffer.

2.5.4 Configure syntax checkers

Many syntax checkers provide command line flags to change their behaviour. Flycheck wraps important flags as regular Emacs user options.

The *list of supported languages* includes all options for each syntax checker. You can change these options in the Customize interface under *programming* → *tools* → *flycheck* → *flycheck-options*, however we recommend to use Directory Variables to configure syntax checkers per project.

See also:

Directory Variables(emacs) Information about directory variables.

Configuration files

Some syntax checkers can additionally read configuration from files. Flycheck can find configuration files of syntax checkers and use them when invoking the syntax checker program:

defcustom flycheck-local-config-file-functions
Functions to call to find a configuration file for a syntax checker. Each function gets the name of a configuration file and shall return the absolute path to a file if one exists. The default value leads to the following steps:

1. If the name is an absolute path, use it.
2. If the name exists in any ancestor directory, use the nearest one.
3. If the name exists in `$HOME`, use it.

This option is an abnormal hook, see *Hooks(elisp)*.

Flycheck takes the names of configuration files from user options defined for syntax checkers that support configuration files. Like above the *list of languages* also lists all supported configuration file options. You can also change these in Customize, under *programming* → *tools* → *flycheck* → *flycheck-config-files*, but again we recommend to use Directory Variables.

We also recommend to prefer configuration files over options as you can usually commit the configuration files to your source control repository to share them with other contributors so that all contributors can use the same configuration for syntax checking and linting.

2.5.5 Change syntax checker executables

Flycheck normally tries to run syntax checker tools by their standard name from `exec-path`. Sometimes, though, you need to use a different version of a tool, or probably don't even have a tool available globally—this frequently occurs in Javascript project where dependencies including linter tools are typically installed into a local `node_modules` directory:

`M-x flycheck-set-checker-executable`
Prompt for a syntax checker and an executable file and make Flycheck use the executable file for the syntax checker in the current buffer.

Internally this command sets a variable named `flycheck-`*`checker`*`-executable` where *`checker`* is the name of the syntax checker entered on the prompt, e.g. *`c/c++-clang`*.

Flycheck defines these *executable options* for every syntax checker that runs an external command. You can change these variables with directory variables or set them in custom Emacs Lisp code such as mode hooks.

See also:

Directory Variables(emacs) Information about directory variables.

2.5.6 Configuring checker chains

In any given buffer where Flycheck is enabled, only one checker may be run at a time. However, any number of checkers can be run in sequence. In such a sequence, after the first checker has finished running and its errors have been reported, the next checker of the sequence runs and its errors are reported, etc. until there are no more checkers in the sequence. This sequence is called a *checker chain*.

Some checkers chains are already setup by default in Flycheck: e.g., *`emacs-lisp`* will be followed by *`emacs-lisp-checkdoc`*, and *`python-mypy`* will be followed by *`python-flake8`*.

When defining a checker, you can specify which checkers may run after it by setting the `:next-checkers` property (see the docstring of `flycheck-define-generic-checker`).

For a given checker, several next checkers may be specified. Flycheck will run the first (in order of declaration) whose error level matches (see below) and which can be used in the current buffer.

You can also customize the next checker property by calling *`flycheck-add-next-checker`* in your Emacs configuration file.

`defun flycheck-add-next-checker` `checker next` **`&optional`** `append`
Set *next* to run after *checker*. Both arguments are syntax checker symbols.

For example, the following will make *`python-pylint`* run after *`python-flake8`*:

```
(flycheck-add-next-checker 'python-flake8 'python-pylint)
```

Next may also be a cons cell `(level . next-checker)`, where *next-checker* is a symbol denoting the syntax checker to run after *checker*, and *level* is an error level. The *next-checker* will then only be run if there is no current error whose level is more severe than *level*. If *level* is `t`, then *next-checker* is run regardless of the current errors.

For instance, if you wanted to run *python-pylint* only if *python-flake8* produced no errors (only warnings and info diagnostics), then you would rather use:

```
(flycheck-add-next-checker 'python-flake8 '(warning . python-pylint))
```

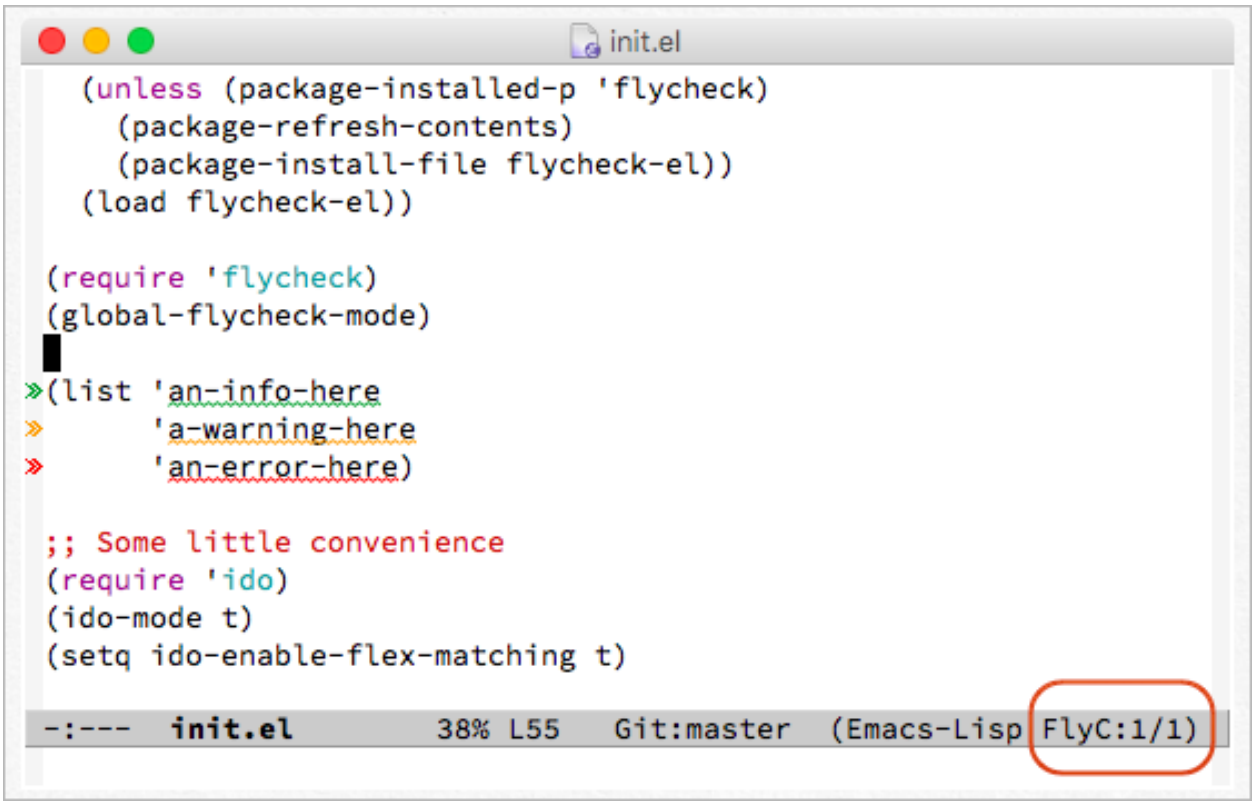
2.6 See errors in buffers

When a syntax check in the current buffer has finished Flycheck reports the results of the check in the current buffer in two ways:

- Highlight errors, warnings, etc. directly in the buffer according to *flycheck-highlighting-mode* and *flycheck-highlighting-style*.
- Indicate errors, warnings, etc. in the fringe according to *flycheck-indication-mode*.

Additionally Flycheck indicates its current state and the number of errors and warnings in the mode line.

The following screenshot illustrates how this looks like in the default Emacs color theme. It shows an info, a warning and an error annotation, from top to bottom. Please also note the fringe indicators on the left side and the emphasized mode line indicator in the bottom right corner:



Note: The colours of fringe icons and the whole appearance of the error highlights depend on the active color theme. Although red, orange and green or blue seem to be somewhat standard colours for Flycheck's annotations across many

popular themes, please take a closer look at your color theme if you're in doubt about the meaning of a Flycheck highlight.

2.6.1 Error levels

All errors that syntax checkers report have a *level* which tells you the severity of the error. Flycheck has three built-in levels:

error Severe errors like syntax or type errors.

warning Potential but not fatal mistakes which you should likely fix nonetheless.

info Purely informational messages which inform about notable things in the current buffer, or provide additional help to fix errors or warnings.

Each error level has a distinct highlighting and colour which helps you to identify the severity of each error right in the buffer.

2.6.2 Error highlights

Flycheck highlights errors directly in the buffer according to *flycheck-highlighting-mode* and *flycheck-highlighting-style*.

Most checkers report a single error position, not a range, so Flycheck typically needs to guess how far to extend the highlighting: by default, it highlights the whole symbol at the location reported by the checker, as in the screenshot above, but you can change that range (or even disable highlighting completely) using *flycheck-highlighting-mode*.

defcustom flycheck-highlighting-mode
How Flycheck chooses which buffer region to highlight:

nil Do not highlight anything at all.

lines Highlight the whole line and discard any information about the column.

columns Highlight the column of the error if any, otherwise like `lines`.

symbols Highlight the entire symbol around the error column if any, otherwise like `columns`. This is this default.

sexps Highlight the entire expression around the error column if any, otherwise like `columns`.

> **Warning:** In some major modes `sexps` is *very* slow, because discovering expression boundaries is costly.
>
> The built-in `python-mode` is known to suffer from this issue.
>
> Be careful when enabling this mode.

Conversely, when a checker reports a range, Flycheck uses that.

The style of the highlighting is determined by the value of *flycheck-highlighting-style*. By default, Flycheck highlights error text with a face indicating the severity of the error (typically, this face applies a coloured wavy underline). Instead of faces, however, Flycheck can also indicate erroneous text by inserting delimiters around it (checkers sometimes report errors that span a large region of the buffer, making underlines distracting, so in fact Flycheck only applies a face if the error spans less than 5 lines; this is achieved using the `conditional` style described below).

defcustom flycheck-highlighting-style
How Flycheck highlights error regions.

nil Do not indicate error regions.

level-face Apply a face to erroneous text.

(delimiters BEFORE AFTER) Bracket the error text between `BEFORE` and `AFTER`, which can be strings, images, etc. Chars are handled specially: they are repeated twice to form double brackets.

(conditional NLINES S1 S2) Chose between styles `S1` and `S2`: `S1` if the error covers up to `NLINES`, and `S2` otherwise.

To change the style of the underline or use different colours in the `level-face` style, customize the following faces, which are used depending on the error level:

defface flycheck-error
defface flycheck-warning
defface flycheck-info
The highlighting face for `error`, `warning` and `info` levels respectively.

Delimiters use the same faces as the fringe icons described below, in addition to the *flycheck-error-delimiter* face; delimited text has the *flycheck-delimited-error* face, which is empty by default.

defface flycheck-error-delimiter
The face applied to `BEFORE` and `AFTER` delimiters.

defface flycheck-delimited-error
The face applied to error text in `delimiters` style.

2.6.3 Fringe and margin icons

In GUI frames, Flycheck also adds indicators to the fringe—the left or right border of an Emacs window—to help you identify erroneous lines quickly. These indicators consist of a rightward-pointing double arrow shape coloured in the colour of the corresponding error level. By default the arrow is 8 pixels wide, but a 16 pixels version is used if the fringe is wide enough.

Note: Flycheck extensions can define custom error levels with different fringe indicators. Furthermore some Emacs distributions like Spacemacs redefine Flycheck's error levels to use different indicators. If you're using such a distribution please take a look at its documentation if you're unsure about the appearance of Flycheck's indicators.

You can customise the location of these indicators (left or right fringe) with *flycheck-indication-mode*, which also lets you turn off these indicators completely; additionally, you can move these indicators into the margins instead of the fringes:

defcustom flycheck-indication-mode
How Flycheck indicates errors and warnings in the buffer fringes:

left-fringe or right-fringe Use the left or right fringe respectively. Fringes can only contain monochrome bitmaps, so Flycheck draws small pixel-art arrows.

left-margin or right-margin Use the left or right margin respectively. Margins can support all of Emacs' rendering facilities, so Flycheck uses the » character, which scales with the font size.

nil Do not indicate errors and warnings in the fringe or in the margin.

By default, Emacs displays fringes, but not margins. With `left-margin` and `right-margin` indication modes, you will need to enable margins in your `.emacs`. For example:

```
(setq-default left-fringe-width 1 right-fringe-width 8
              left-margin-width 1 right-margin-width 0)
```

If you intend to use margins only with Flycheck, consider using `flycheck-set-indication-mode` in a hook instead; this function adjusts margins and fringes for the current buffer.

```
(setq-default flycheck-indication-mode 'left-margin)
(add-hook 'flycheck-mode-hook #'flycheck-set-indication-mode)
```

That function sets fringes and margins to reasonable (but opinionated) defaults, according to `flycheck-indication-mode`. To set your own margin and fringe widths, use a hook and call `flycheck-refresh-fringes-and-margins`, like this:

```
;; Show indicators in the left margin
(setq flycheck-indication-mode 'left-margin)

;; Adjust margins and fringe widths...
(defun my/set-flycheck-margins ()
  (setq left-fringe-width 8 right-fringe-width 8
        left-margin-width 1 right-margin-width 0)
  (flycheck-refresh-fringes-and-margins))

;; ...every time Flycheck is activated in a new buffer
(add-hook 'flycheck-mode-hook #'my/set-flycheck-margins)
```

The following faces control the colours of fringe and margin indicators.

defface flycheck-fringe-error
defface flycheck-fringe-warning
defface flycheck-fringe-info
The icon faces for `error`, `warning` and `info` levels respectively.

When an error spans multiple lines, Flycheck displays a hatch pattern in the fringes or vertical dots in the margins to indicate the extent of the error.

To change the fringe bitmap or the symbol used in the margins, use the function `flycheck-redefine-standard-error-levels`.

2.6.4 Mode line

Like all minor modes Flycheck also has a mode line indicator. You can see it in the bottom right corner of the above screenshot. By default the indicator shows Flycheck's current state via one of the following texts:

`FlyC*`	Flycheck is checking the buffer currently.
`FlyC`	There are no errors or warnings in the current buffer.
`FlyC:3/5`	There are three errors and five warnings in the current buffer.
`FlyC-`	Flycheck did not find a syntax checker for the current buffer. Take a look at the *list of supported languages* and type `C-c ! v` to see what checkers are available for the current buffer.
`FlyC!`	The last syntax check failed. Inspect the `*Messages*` buffer look for error messages, and consider *reporting a bug*.
`FlyC?`	The last syntax check had a dubious result. The definition of a syntax checker may have a bug. Inspect the `*Messages*` buffer and consider *reporting a bug*.

You can entirely customise the mode line indicator with `flycheck-mode-line`:

defcustom flycheck-mode-line
A "mode line construct" for Flycheck's mode line indicator.

See also:

Mode Line Data(elisp) Documentation of mode line constructs.

flycheck-status-emoji A Flycheck extension which puts emojis into Flycheck's mode line indicator.

flycheck-color-mode-line A Flycheck extension which colours the entire mode line according to Flycheck's status.

2.6.5 Error thresholds

To avoid flooding a buffers with excessive highlighting, cluttering the appearance and slowing down Emacs, Flycheck takes precautions against syntax checkers that report a large number of errors exceeding *flycheck-checker-error-threshold*:

defcustom flycheck-checker-error-threshold
The maximum number of errors a syntax checker is allowed to report.

If a syntax checker reports more errors the error information is **discarded**. To not run into the same issue again on the next syntax check the syntax checker is automatically added to *flycheck-disabled-checkers* in this case to disable it for the next syntax check.

2.6.6 Clear results

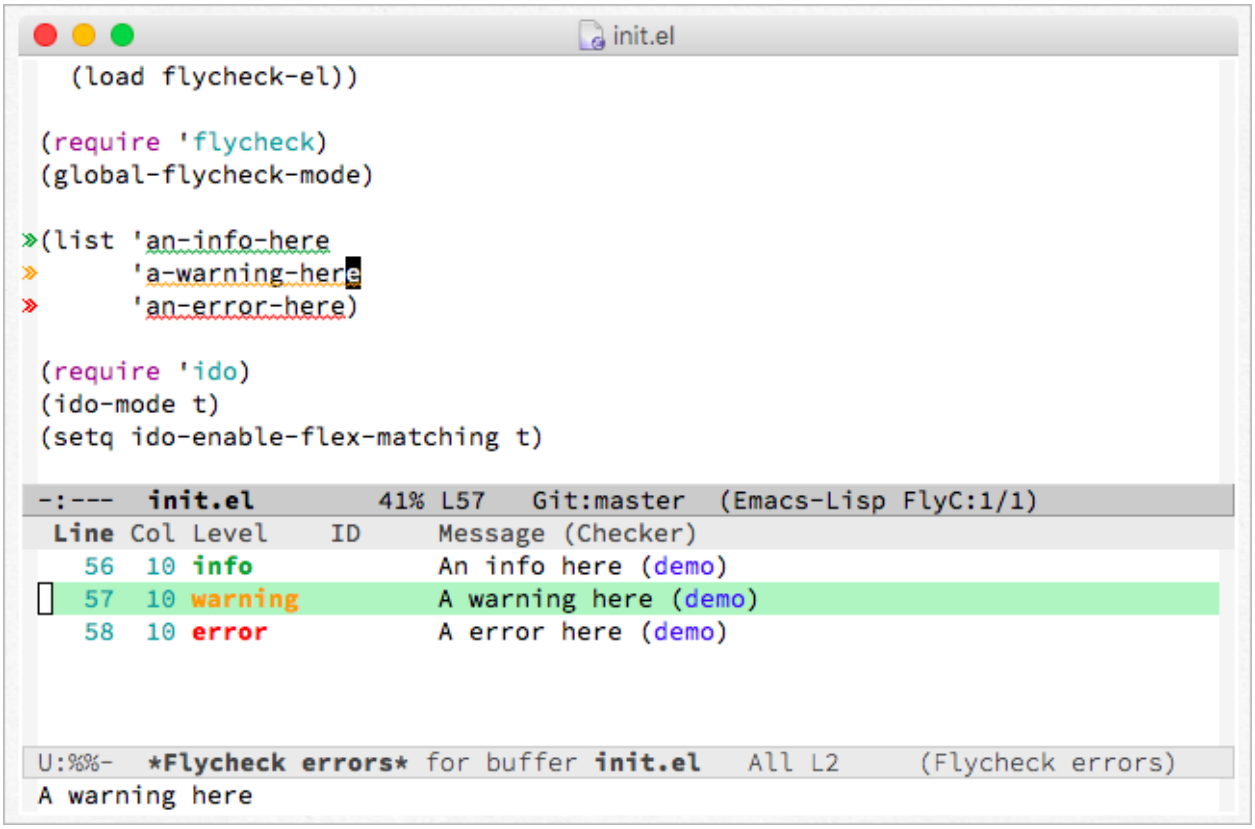
You can explicitly remove all highlighting and indication and all error information from a buffer:

C-c ! C
M-x flycheck-clear
Clear all reported errors, all highlighting and all indication icons from the current buffer.

C-u C-c ! C
C-u M-x flycheck-clear
Like *C-c ! C* but also interrupt any syntax check currently running. Use this command if you think that Flycheck is stuck.

2.7 List all errors

You can see all errors in the current buffer in Flycheck's error list:



The key `C-c ! l` pops up the error list:

C-c ! l
M-x flycheck-list-errors
M-x list-flycheck-errors
Pop up a list of errors in the current buffer.

The error list automatically updates itself after every syntax check and follows the current buffer: If you switch to different buffer or window it automatically shows the errors of the now current buffer. The buffer whose errors are shown in the error list is the *source buffer*.

Whenever the point is on an error in the *source buffer* the error list highlights these errors—the green line in the screenshot above.

Within the error list the following key bindings are available:

RET	Go to the current error in the source buffer
n	Jump to the next error
p	Jump to the previous error
e	Explain the error
f	Filter the error list by level
F	Remove the filter
S	Sort the error list by the column at point
g	Check the source buffer and update the error list
q	Quit the error list and hide its window

2.7.1 Filter the list

By the default the error list shows all errors but sometimes you'd like to hide warnings to focus only on real errors. The error list lets you hide all errors below a certain level with `f`. This key prompts for an error level and will remove all errors of lower levels from the list. The filter is permanent as long as the error list buffer stays alive or the filter is reset with `F`.

2.7.2 Sort the list

You can press `S` or click on the column headings to sort the error list by any of the following columns:

- Line
- Level
- ID
- Message and checker

Click twice or press `S` repeatedly to flip the sort order from ascending to descending or vice versa.

2.7.3 Tune error list display

By default the error list buffer pops up like any other buffer. Flycheck does not enforce special rules on how it's displayed and where it's located in the frame so essentially the error list pops up at arbitrary places wherever Emacs can find a window for it.

However you can tell Emacs to obey certain rules when displaying buffers by customizing the built-in option `display-buffer-alist`. You can use this option to make the error list display like similar lists in contemporary IDEs like VisualStudio, Eclipse, etc. with the following code in your *init file*:

```
(add-to-list 'display-buffer-alist
             `(,(rx bos "*Flycheck errors*" eos)
              (display-buffer-reuse-window
               display-buffer-in-side-window)
              (side            . bottom)
              (reusable-frames . visible)
              (window-height   . 0.33)))
```

This display rule tells Emacs to always display the error list at the bottom side of the frame, occupying a third of the entire height of the frame.

See also:

Shackle An Emacs package which provides an alternative way to control buffer display

2.8 Interact with errors

There are a couple of things that you can do with Flycheck errors in a buffer:

- You can navigate to errors, and go to the next or previous error.
- You can display errors to read their error messages.
- You can put error messages and IDs into the kill ring.

This section documents the corresponding commands and their customisation options.

2.8.1 Navigate errors

By default Flycheck hooks into Emacs' standard error navigation on `M-g n` (`next-error`) and `M-g p` (`previous-error`). When *`Flycheck Mode`* is enabled these commands will jump to the next and previous Flycheck error respectively. See *Compilation Mode(emacs)* for more information about these commands.

This way you don't need to learn special keybindings to navigate Flycheck errors; navigation should just work out of the box.

Note: Visible compilation buffers such as buffers from `M-x compile`, `M-x grep`, etc. still take *precedence* over Flycheck's errors.

The exact behaviour of these error navigation features is very context-dependent and can be very confusing at times so you can disable this integration:

`defcustom flycheck-standard-error-navigation`
Whether to integrate Flycheck errors into Emacs' standard error navigation. Defaults to `t`, set to `nil` to disable.

Important: When changing the value you must disable *`Flycheck Mode`* and enable it again for the change to take effect in any buffers where *`Flycheck Mode`* is enabled.

Flycheck provides an independent set of navigation commands which will always navigate Flycheck errors in the current buffer, regardless of visible compilation buffers and *`flycheck-standard-error-navigation`*:

`C-c ! n`
`M-x flycheck-next-error`
Jump to the next error.

With prefix argument jump forwards by as many errors as specified by the prefix argument, e.g. `M-3 C-c ! n` will move to the 3rd error from the current point. With negative prefix argument move to previous errors instead. Signal an error if there are no more Flycheck errors.

`C-c ! p`
`M-x flycheck-previous-error`
Jump to the previous Flycheck error.

With prefix argument jump backwards by as many errors as specified by the prefix argument, e.g. `M-3 C-c ! p` will move to the 3rd error before the current point. With negative prefix argument move to next errors instead. Signal an error if there are no more Flycheck errors.

`M-x flycheck-first-error`
Jump to the first Flycheck error.

With prefix argument, jump forwards to by as many errors as specified by the prefix argument, e.g. `M-3 M-x flycheck-first-error` moves to the 3rd error from the beginning of the buffer. With negative prefix argument move to the last error instead.

By default error navigation jumps to all errors but you can choose to skip over errors with low levels:

`defcustom flycheck-navigation-minimum-level`
The minimum levels of errors to consider for navigation.

If set to an error level only navigate to errors whose level is as least as severe as this one. If `nil` navigate to all errors.

2.8.2 Display errors

Whenever you move point to an error location Flycheck automatically displays all Flycheck errors at point after a short delay which you can customise:

defcustom flycheck-display-errors-delay
The number of seconds to wait before displaying the error at point. Floating point numbers can express fractions of seconds.

By default Flycheck shows the error messages in the minibuffer or in a separate buffer if the minibuffer is too small to hold the whole error message but this behaviour is entirely customisable:

defcustom flycheck-display-errors-function
A function to display errors.

The function is given the list of Flycheck errors to display as sole argument and shall display these errors to the user in some way.

Flycheck provides two built-in functions for this option:

defun flycheck-display-error-messages errors
defun flycheck-display-error-messages-unless-error-list errors
Show error messages and IDs in the echo area or in a separate buffer if the echo area is too small (using `display-message-or-buffer` which see). The latter only displays errors when the *error list* is not visible. To enable it add the following to your *init file*:

```
(setq flycheck-display-errors-function
      #'flycheck-display-error-messages-unless-error-list)
```

See also:

flycheck-pos-tip A Flycheck extension to display errors in a GUI popup.

Additionally Flycheck shows errors in a GUI tooltip whenever you hover an error location with the mouse pointer. By default the tooltip contains the messages and IDs of all errors under the pointer, but the contents are customisable:

defcustom flycheck-help-echo-function
A function to create the contents of the tooltip.

The function is given a list of Flycheck errors to display as sole argument and shall return a single string to use as the contents of the tooltip.

2.8.3 Errors from other files

Some checkers may return errors from files other than the current buffer (e.g., `gcc` may complain about errors in included files). These errors appear in the error list, and are also added on the first line of the current buffer. You can jump to the incriminating files with *`flycheck-previous-error`*.

By default, warnings and info messages from other files are ignored, but you can customize the minimum level:

defcustom flycheck-relevant-error-other-file-minimum-level
The minimum level errors from other files to consider for inclusion in the current buffer.

If set to an error level, only display errors from other files whose error level is at least as severe as this one. If `nil`, display all errors from other files.

To never show any errors from other files, set *`flycheck-relevant-error-other-file-show`* to `nil`.

defcustom flycheck-relevant-error-other-file-show
Whether to show errors from other files.

2.8.4 Explain errors

Flycheck also has the ability to display explanations for errors, provided the error checker is capable of producing these explanations. Currently, only the *rust* and *rust-cargo* checkers produce explanations.

C-c ! e
M-x flycheck-explain-error-at-point
Display an explanation for the first explainable error at point.

2.8.5 Kill errors

You can put errors into the kill ring with C-c ! w:

C-c ! C-w
M-x flycheck-copy-errors-as-kill
Copy all messages of the errors at point into the kill ring.

C-u C-c ! C-w
C-u M-x flycheck-copy-errors-as-kill
Like C-c ! w but with error IDs.

M-0 C-c ! C-w
M-0 M-x flycheck-copy-errors-as-kill
Like C-c ! w but do not copy the error messages but only the error IDs.

2.9 Flycheck versus Flymake

This article compares Flycheck to the *built-in* Flymake mode. It does not consider third-party extensions such as flymake-easy, but references them at appropriate places.

We aim for this comparison to be fair and comprehensive, but it may contain stale information. Please report any inaccuracy you might find, and feel free to edit this page and improve it.

Note: This comparison was updated at the time of the Emacs 26.1 release, which contains an overhaul of Flymake. If you are using Emacs 25.3 or below, you can still access the comparison between Flycheck and the legacy Flymake here.

2.9.1 Overview

This table gives an overview of the differences and similarities between Flycheck and Flymake. The rest of this page describes each point in more detail.

	Flycheck	Flymake
Supports Emacs versions	24.3+	26.1+
Built-in	no	yes
Supported languages	100+ built-in, 200+ w/ 3rd-party	10 built-in, 50+ w/ 3rd party
Automatic syntax checking	built-in	manual
Check triggers	save, newline, change, buffer switch	save, newline, change
Asynchronous checking	yes, always	yes, for some modes
Automatic syntax checker selection	by major mode and custom predicates	no
Multiple syntax checkers per buffer	yes (configurable chain)	yes (all at once)
Definition of new syntax checkers	single declarative macro	arbitrary function[1]
Configuration debugging	built-in (C-c ! v)	none
Error identifiers	yes	no
Error explanations	yes	no
Error parsing helpers	for regexp, JSON and XML	none
Fringe icons for errors	yes	yes
Error highlighting	faces, brackets, mixed	faces only
Error indicators	fringes (incl HiDPI), margins	fringes only
Error message display	tooltip, echo area, fully customizable (e.g. tooltip, popup w/ 3rd party packages)	tooltip, echo area
List of all errors	yes; filterable by error level	yes

2.9.2 Detailed review

Relation to Emacs

Flymake has been part of GNU Emacs since GNU Emacs 22. As such, contributions to Flymake are subject to the FSF policies on GNU projects. Most notably, contributors are required to assign their copyright to the FSF.

Flycheck is not part of GNU Emacs. However, it is free software as well, and publicly developed on the well-known code hosting platform Github. Contributing to Flycheck does not require a copyright assignment, only an explicit agreement that your contributions will be licensed under the GPL.

Automatic syntax checking

Flymake is not enabled automatically for supported languages. It must be enabled for each mode individually, or by, e.g., adding to a hook that enables it for all `prog-mode` buffers. If no backends for the major mode are available, Flymake will non-intrusively tell you in the modeline.

Flycheck provides a global mode *`global-flycheck-mode`* which enables syntax checking in every supported language, where it is safe to do so (remote and encrypted buffers are excluded by default).

Syntax checkers

[1] flymake-easy provides a function to define a new syntax checker, which sets all required variables at once.

Supported languages

Flymake comes with support for Emacs Lisp, Ruby (`ruby` for syntax check and `rubocop` for lints), Python and Perl. In addition, backends written for the legacy Flymake are compatible with the new implementation.

Flycheck provides support for *over 50 languages* with over 100 syntax checkers, most of them contributed by the community.

Definition of new syntax checkers

Flymake backends are single functions which report diagnostics to a callback function given as argument.

Flycheck provides a single function `flycheck-define-checker` to define a new syntax checker. This function uses a declarative syntax which is easy to understand even for users unfamiliar with Emacs Lisp. In fact, most syntax checkers in Flycheck were contributed by the community.

For example, the Perl checker in Flycheck is defined as follows:

```
(flycheck-define-checker perl
  "A Perl syntax checker using the Perl interpreter.

See URL `http://www.perl.org'."
  :command ("perl" "-w" "-c" source)
  :error-patterns
  ((error line-start (minimal-match (message))
          " at " (file-name) " line " line
          (or "." (and ", " (zero-or-more not-newline))) line-end))
  :modes (perl-mode cperl-mode))
```

The whole process is described in *Adding a syntax checker to Flycheck*.

Customization of syntax checkers

Flymake does not provide built-in means to customize syntax checkers. Instead, when defining a new syntax checker the user needs to declare customization variables explicitly and check their value in the init function.

Flycheck provides built-in functions to add customization variables to syntax checkers and splice the value of these variables into the argument list of a syntax checking tool. Many syntax checkers in Flycheck provide customization variables. For instance, you can customize the enabled warnings for C with *`flycheck-clang-warnings`*. Flycheck also tries to automatically find configuration files for syntax checkers.

Executables of syntax checkers

Flymake does not provide built-in means to change the executable of a syntax checker.

Flycheck defines a variable to set the path of a syntax checker tool for each defined syntax checker and provides the interactive command *`flycheck-set-checker-executable`* to change the executable used in a buffer. The process used to locate checker configuration files can also be customized using `flycheck-locate-config-file-functions`, allowing you to store your personal checker configuration files in your `.emacs.d` folder.

Syntax checker selection

Flymake runs all functions added to the `flymake-diagnostic-functions` hook.

Flycheck uses the major mode and checker-specific predicates to automatically select a syntax checker.

Custom predicates

Flymake may allow for backends to implement custom logic to decide whether to run the check or not. There are no easily-defined predicate functions.

Flycheck supports custom predicate functions. For instance, Emacs uses a single major mode for various shell script types (e.g. Bash, Zsh, POSIX Shell, etc.), so Flycheck additionally uses a custom predicate to look at the value of the variable `sh-shell` in Sh Mode buffers to determine which shell to use for syntax checking.

Manual selection

Flymake users may manually select a specific backend by overriding the value of the backends list.

Flycheck provides the local variable *`flycheck-checker`* to explicitly use a specific syntax checker for a buffer and the command *`flycheck-select-checker`* to set this variable interactively.

Multiple syntax checkers per buffer

Flymake will use all the backends added to the `flymake-diagnostic-functions` hook to check a buffer; all backends are started at the same time, but errors are reported in the buffer as soon as a backend returns them. Backends can also be written to first report errors for the visible region of the buffer, and collect errors for hidden regions later.

Flycheck can also apply multiple syntax checkers per buffer, but checkers run in sequence rather than concurrently. For instance, Flycheck will check PHP files with PHP CLI first to find syntax errors, then with PHP MessDetector to additionally find idiomatic and semantic errors, and eventually with PHP CheckStyle to find stylistic errors. The user will see all errors reported by all of these tools in the buffer. These checker-chains are configurable (see *Configuring checker chains*), so it's possible to run an advanced style checker only if a basic syntax checker returned no errors (this avoids accumulating too many false positives and improves performance).

Errors

Error identifiers

Flymake does not include special treatment for error identifiers.

Flycheck supports identifiers for different kinds of errors, if a syntax checker provides these. The identifiers appear in the error list and in error display, and can be copied independently, for instance for use in an inline suppression comment or to search the web for a particular kind of error.

Error explanations

Some **Flycheck** checkers can use error identifiers to provide error explanations in an help buffer (see *`flycheck-explain-error-at-point`*).

Error indicators

Both **Flymake** and **Flycheck** indicate errors in the buffer (using overlays) and in the fringes. Flycheck includes fringe bitmaps for HiDPI screens, and also supports displaying indicators in the margins instead of the fringes (this behavior can be customized using *flycheck-indication-mode*, and *flycheck-highlighting-mode*).

Error parsing

Flymake lets backend parse error messages from tools. There are no built-in helpers for defining error patterns, or for parsing JSON or XML formats.

Flycheck checkers can use regular expressions as well as custom parsing functions. The preferred way to define a checker is to use the `rx` syntax, extended with custom forms for readable error patterns. Flycheck includes some ready-to-use parsing functions for common output formats, such as Checkstyle XML, or JSON interleaved with plain text.

Error message display

Flymake shows error messages in a tool tip if the user hovers the mouse over an error location, or in the echo area if the user navigates to the error with `flymake-goto-next-error`.

Flycheck shows error message in tool tips as well, and also displays error messages in the echo area if the point is at an error location. This feature is fully customizable via *flycheck-display-errors-function*, and several *extensions* already provide alternative way to display errors.

CHAPTER 3

The Community Guide

The Community Guide provides information about Flycheck's ecosystem and community.

3.1 Flycheck Code of Conduct

Our Code of Conduct defines the social norms and policies within Flycheck's community. Whenever you interact with Flycheck or Flycheck developers, whether in our official channels or privately, you're expected to follow this Code of Conduct.

3.1.1 Conduct

Contact: *Any moderator*

- We are committed to providing a friendly, safe and welcoming environment for all, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, religion, nationality, or similar personal characteristic.
- Please avoid using overtly sexual nicknames or other nicknames that might detract from a friendly, safe and welcoming environment for all.
- Please be kind and courteous. There's no need to be mean or rude.
- Please do not curse or use bad words. Foul language will not help us to build a great product.
- Respect that people have differences of opinion and that every design or implementation choice carries a trade-off and numerous costs. There is seldom a right answer.
- Please keep unstructured critique to a minimum. If you have solid ideas you want to experiment with, make a fork and see how it works.
- We will exclude you from interaction if you insult, demean or harass anyone. That is not welcome behaviour. We interpret the term "harassment" as including the definition in the Citizen Code of Conduct; if you have any lack of clarity about what might be included in that concept, please read their definition. In particular, we don't tolerate behavior that excludes people in socially marginalized groups.

- Private harassment is also unacceptable. No matter who you are, if you feel you have been or are being harassed or made uncomfortable by a community member, please contact a *moderator* immediately. Whether you're a regular contributor or a newcomer, we care about making this community a safe place for you and we've got your back.
- Likewise any spamming, trolling, flaming, baiting or other attention-stealing behaviour is not welcome.

3.1.2 Moderation

These are the policies for upholding our community's standards of conduct in our communication channels, most notably in Flycheck's Github organisation and in Flycheck's Gitter channels.

1. Remarks that violate the Flycheck code of conduct, including hateful, hurtful, oppressive, or exclusionary remarks, are not allowed.
2. Remarks that moderators find inappropriate, whether listed in the code of conduct or not, are also not allowed.
3. Moderators will first respond to such remarks with a warning.
4. If the warning is unheeded, the user will be "kicked," i.e., kicked out of the communication channel to cool off.
5. If the user comes back and continues to make trouble, they will be banned, i.e., indefinitely excluded.
6. Moderators may choose at their discretion to un-ban the user if it was a first offense and they offer the offended party a genuine apology.
7. If a moderator bans someone and you think it was unjustified, please take it up with that moderator, or with a different moderator, **in private**. Complaints about bans in-channel are not allowed.
8. Moderators are held to a higher standard than other community members. If a moderator creates an inappropriate situation, they should expect less leeway than others.

In the Flycheck community we strive to go the extra step to look out for each other. Don't just aim to be technically unimpeachable, try to be your best self. In particular, avoid flirting with offensive or sensitive issues, particularly if they're off-topic; this all too often leads to unnecessary fights, hurt feelings, and damaged trust; worse, it can drive people away from the community entirely.

And if someone takes issue with something you said or did, resist the urge to be defensive. Just stop doing what it was they complained about and apologize. Even if you feel you were misinterpreted or unfairly accused, chances are good there was something you could have communicated better — remember that it's your responsibility to make your fellow Flycheck people comfortable. Everyone wants to get along and we are all here first and foremost because we want to talk about cool technology. You will find that people will be eager to assume good intent and forgive as long as you earn their trust.

—

Adapted from the Rust Code of Conduct.

3.2 Recommended extensions

The Emacs community has produced a number of extensions to Flycheck. This page lists all that we know of and can safely recommend to our users.

Official extensions are (co-)maintained by the *Flycheck maintainers* who will take care to update official extensions in case of breaking changes in Flycheck and work to provide extra API for extensions if needed. If you'd like to make your extension an *official* one and move it into the Flycheck Github organisation please contact a *maintainer*.

If you do know extensions not in this list, or would like to see your own extension here, please feel free to add it.

We would like to thank all people who created and contributed to Flycheck extensions for their awesome work. Without your help and support Flycheck would not be what it is today.

3.2.1 User interface

These extensions change Flycheck's user interface:

- flycheck-color-mode-line (*official*) colors the mode line according to the Flycheck status.
- flycheck-pos-tip (*official*) shows Flycheck error messages in a graphical popup.
- liblit/flycheck-status-emoji adds cute emoji (e.g. for errors) to Flycheck's mode line status.
- Wilfred/flycheck-title shows Flycheck error messages in the frame title.
- flycheck-inline shows Flycheck error messages in the buffer, directly below their origin.

3.2.2 Language support

These extensions add support for new languages, or improve support for built-in languages. They are grouped by the corresponding language so you can jump directly to the languages that interest you:

Languages

- *Cadence*
- *Clojure*
- *C/C++/Objective C*
- *D*
- *Elixir*
- *Emacs Lisp*
- *Julia*
- *Haskell*
- *Ledger*
- *Mercury*
- *OCaml*
- *PHP*
- *Python*
- *Rust*
- *Shell scripts*

Cadence

- cmarqu/flycheck-hdl-irun adds a syntax checker for hardware description languages supported by Cadence IES/irun.

Clojure

- clojure-emacs/squiggly-clojure adds syntax checking for Clojure.

C/C++/Objective C

- Wilfred/flycheck-pkg-config configures Flycheck to use settings from pkg-config when checking C/C++.
- Sarcasm/flycheck-irony adds a Flycheck syntax checker for C, C++ and Objective C using Irony Mode.

D

- flycheck-d-unittest (*official*) adds a Flycheck checker to run unit tests for D programs on the fly.

Elixir

- tomekowal/flycheck-mix adds an Elixir syntax checker using the `mix` build tool.

Emacs Lisp

- flycheck-cask (*official*) makes Flycheck use Cask packages for Emacs Lisp syntax checking in Cask projects.
- purcell/flycheck-package checks Emacs Lisp packages for common problems with package metadata.

Julia

- gdkrmr/flycheck-julia makes linting for Julia available via Lint.jl.

Haskell

- flycheck-haskell (*official*) configures Flycheck from the Cabal settings and sandbox in Haskell projects.

Ledger

- purcell/flycheck-ledger adds a syntax checker for the Ledger accounting tool.

Mercury

- flycheck-mercury (*official*) adds a syntax checker for the Mercury language.

OCaml

- flycheck-ocaml (*official*) adds a syntax checker for OCaml using the Merlin backend.

PHP

- emacs-php/phpstan.el adds a PHP static analyzer using PHPStan.
- emacs-php/psalm.el adds a PHP static analyzer using Psalm.

Python

- Wilfred/flycheck-pyflakes adds a Python syntax checker using Pyflakes.
- msherry/flycheck-pycheckers adds a checker for Python that can run multiple syntax checkers simultaneously (Pyflakes, PEP8, Mypy 2/3, etc.).
- chocoelho/flycheck-prospector adds Prospector checker for Python syntax.

Rust

- flycheck-rust (*official*) configures Flycheck according to the Cargo settings and layouts of the current Rust project.

Shell scripts

- Gnouc/flycheck-checkbashisms adds a shell script syntax checker using `checkbashisms` which is part of Debian devscripts and checks for common Bash constructs in POSIX shell scripts.

3.3 Get help

Please ask questions about Flycheck on Stack Exchange or in our Gitter chat. We try to answer all questions as fast and as precise as possible.

To report bugs and problems please please use our issue tracker. Please note that we have a special policy for *Windows-only issues*.

Please follow our *Code of Conduct* in all these places.

3.4 People

3.4.1 Teams

Maintainers

- **Clément Pit-Claudel** (cpitclaudel, owner)
- **fmdkdd** (fmdkdd, owner)

We maintain Flycheck and all official extensions within the Flycheck organisation, and set the direction and scope of Flycheck. We review and accept pull requests and feature proposals and fix bugs in Flycheck.

Emphasized users are also owners of the Flycheck Organisation, and thus have administrative privileges for all repositories in Flycheck. Notably only owners can currently make Flycheck releases, and their GPG keys sign release tags for Flycheck.

Mention with `@flycheck/maintainers`.

Moderators

Our moderators help uphold our *Flycheck Code of Conduct*. Currently, we do not have a dedicated moderation team; all our *Maintainers* also serve as moderators in our Github organisation and in our official communication channels.

Mention with `@flycheck/moderators`.

Note: If you'd like to help out with moderation, please contact a maintainer.

Language teams

These teams provide support for particular languages in Flycheck.

Elixir

- Aaron Jensen (aaronjensen)
- Kári Tristan Helgason (kthelgason)

Mention with `@flycheck/elixir`.

Go

- Dominik Honnef (dominikh)

Mention with `@flycheck/go`.

Haskell

- Sergey Vinokurov (sergv)
- Steve Purcell (purcell)

Mention with `@flycheck/haskell`.

Javascript

- Saša Jovanić (Simplify)

Mention with `@flycheck/javascript`.

Lua

- Gordon Gao (ghprince)

Mention with `@flycheck/lua`.

Mercury

- Matthias Güdemann (mgudemann)

Mention with `@flycheck/mercury`.

PHP

- USAMI Kenta (zonuexe)

Mention with `@flycheck/php`.

Puppet

- Romanos Skiadas (rski)

Mention with `@flycheck/puppet`.

Ruby

- Saša Jovanić (Simplify)

Mention with `@flycheck/javascript`.

Rust

- fmdkdd
- Michael Pankov (mkpankov)

Mention with `@flycheck/rust`.

TypeScript

- Saša Jovanić (Simplify)

Mention with `@flycheck/typescript`.

3.4.2 Packagers

We would like to thank all people who package Flycheck on behalf of distributions and support our development efforts with their feedback, their patches and their testing:

- Sean Whitton (spwhitton) and the Debian Emacs addon team (Debian packages)

3.4.3 Acknowledgements

We would also like to thank the following people and projects:

- Sebastian Wiesner (lunaryorn) for creating Flycheck in the first place, for taking the time and dedication to maintain it for over 4 years, while maintaining high standards of code quality and nurturing a healthy, active community around it, giving Flycheck the best chances to thrive after his departure.

- Bozhidar Batsov (bbatsov) for his valuable feedback and his constant support and endorsement of Flycheck from the very beginning. Notably he added Flycheck to his popular Prelude project at a very early stage and thus brought Flycheck to many new users.
- Magnar Sveen (magnars) for his dash.el and s.el libraries, which support considerable parts of Flycheck internals, and greatly helped to overcome Sebastian's initial aversion to Emacs Lisp.
- Martin Grenfell (scrooloose) for the Vim syntax checking extension Syntastic which saved Sebastian's life back when he was using Vim, and served as inspiration for Flycheck and many of its syntax checkers.
- Matthias Güdemann (mgudemann), for his invaluable work on Flycheck's logo.
- Pavel Kobyakov for his work on GNU Flymake, which is a great work on its own, despite its flaws and weaknesses.
- Simon Carter (bbbscarter), for his patient in-depth testing of automatic syntax checking, and his very constructive feedback.
- Steve Purcell (purcell) for his valuable feedback, the fruitful discussions and his important ideas about the shape and design of Flycheck, and his indispensable and dedicated work on MELPA, which drives the continuous distribution of Flycheck to its users.

3.4.4 Contributors

The following people—listed in alphabetical order—contributed substantial code to Flycheck:

- Aaron Jensen (aaronjensen)
- Alain Kalker (ackalker)
- Alex Reed (acr4)
- Atila Neves (atilaneves)
- Ben Sless (bsless)
- Bozhidar Batsov (bbatsov)
- Clément Pit-Claudel (cpitclaudel, maintainer, owner)
- Colin Marquardt (cmarqu)
- Cristian Capdevila (capdevc)
- Damon Haley (dhaley)
- David Caldwell (caldwell)
- David Holm (dholm)
- DEADB17 (DEADB17)
- Deokhwan Kim (dkim)
- Derek Chen-Becker (dchenbecker)
- Derek Harland (donkopotamus)
- Dominik Honnef (dominikh)
- Doug MacEachern (dougm)
- Drew Wells (drewwells)
- Erik Hetzner (egh)
- Fanael Linithien (Fanael)

- fmdkdd (maintainer, owner)
- Fred Morcos (fredmorcos)
- Gereon Frey (gfrey)
- Gordon Gao (ghprince)
- Guido Kraemer (gdkrmr)
- Gulshan Singh (gsingh93)
- Iain Beeston (iainbeeston)
- Ibrahim Awwal (ibrahima)
- Jackson Ray Hamilton (jacksonrayhamilton)
- Jim Hester (jimhester)
- Jimmy Yuen Ho Wong (wyuenho)
- Joe DeVivo (joedevivo)
- John Shahid (jvshahid)
- Juergen Hoetzel (juergenhoetzel)
- Kári Tristan Helgason (kthelgason)
- Krzysztof Witkowski (kwitek)
- Lee Adams (leeaustinadams)
- Loïc Damien (dzamlo)
- Lorenzo Villani (lvillani)
- Łukasz Jędrzejewski (jedrz)
- Magnar Sveen (magnars)
- Malyshev Artem (proofit404)
- Manuel Uberti (manuel-uberti)
- Marc Sherry (msherry)
- Marcin Antczak (marcinant)
- Marcus Majewski (hekto)
- Marian Schubert (maio)
- Mario Rodas (marsam)
- Mark Laws (drvink)
- Mark Hellewell (markhellewell)
- Mark Karpov (mrkkrp)
- Martin Polden (mpolden)
- mats cronqvist (massemanet)
- Matthew Curry (strawhatguy)
- Matthias Dahl (BinaryKhaos)
- Michael Pankov (mkpankov)

- Michael Alan Dorman (mdorman)
- Miles Yucht (mgyucht)
- Miro Bezjak (mbezjak)
- Mitch Tishmack (mitchty)
- Moritz Bunkus (mbunkus)
- Omair Majid (omajid)
- papaeye
- Per Nordlöw (nordlow)
- Peter Eisentraut (petere)
- Peter Hoeg (peterhoeg)
- Peter Oliver (mavit)
- Peter Vasil (ptrv)
- Philipp Stephani (phst)
- Robert Dallas Gray (rdallasgray)
- Robert O'Connor (robbyoconnor)
- Robert Zaremba (robert-zaremba)
- Romano Skiadas (rski)
- Saša Jovanić (Simplify)
- Sean Gillespie (swgillespie)
- Sean Salmon (phatcabbage)
- Sean Whitton (spwhitton)
- Sebastian Beyer (sebastianbeyer)
- Sebastian Wiesner (lunaryorn, founder, former maintainer, former owner)
- Sebastian Schlueppel (s3bs)
- Sergey Vinokurov (sergv)
- Stephen Lewis (stephenjlewis)
- Steve Purcell (purcell)
- Sven Keidel (svenkeidel)
- Sylvain Benner (syl20bnr)
- Sylvain Rousseau (thisirs)
- Syohei Yoshida (syohex)
- Ted Zlatanov (tzz)
- Tom Jakubowski (tomjakubowski)
- Tom Willemse (ryuslash)
- Tomoya Tanjo (tom-tan)
- Troy Hinckley (CeleritasCelery)

- Usami Kenta (zonuexe)
- Victor Deryagin (vderyagin)
- Ville Skyttä (scop)
- Vlatko Basic (vlatkoB)
- Wieland Hoffmann (mineo)
- Wilfred Hughes (Wilfred)
- William Cummings (wcummings)
- William Xu (xwl)
- Yannick Roehlly (yannick1974)
- Yasuyuki Oka (yasuyk)
- Zhuo Yuan (yzprofile)

For a complete list of all code contributors see the Contributor Graph or `git shortlog --summary`.

CHAPTER 4

The Developer Guide

The Developer Guide shows how extend Flycheck and how to write syntax checkers for Flycheck.

4.1 Developer's Guide

So you want to extend Flycheck, but have no idea where to start? This guide will give you an overview of Flycheck internals, and take you through adding a syntax checker to Flycheck.

4.1.1 An overview of Flycheck internals

The goal of Flycheck is to display errors from external checker programs directly in the buffer you are editing. Instead of you manually invoking `make` or the compiler for your favorite language, Flycheck takes care of it for you, collects the errors and displays them right there in the buffer.

How Flycheck works is rather straightforward. Whenever a syntax check is started (see *Check buffers*), the following happens:

1. First, Flycheck runs the external program as an asynchronous process using `start-process`. While this process runs, Flycheck simply accumulates its output.
2. When the process exits, Flycheck parses its output in order to collect the errors. The raw output is turned into a list of `flycheck-error` objects containing, among others, the filename, line, column, message and severity of the error.
3. Flycheck then filters the collected errors to keep only the relevant ones. For instance, errors directed at other files than the one you are editing are discarded. The exact sementics of which errors are relevant is defined in `flycheck-relevant-error-p`.
4. Relevant errors are highlighted by Flycheck in the buffer, according to user preference. By default, each error adds a mark in the fringe at the line it occurs, and underlines the symbol at the position of the error using *overlays*.
5. Finally, Flycheck rebuilds the error list buffer.

Flycheck follows this process for all the *many different syntax checkers* that are provided by default.

Note: Specifically, the above describes the process of *command checkers*, i.e., checkers that run external programs. All the checkers defined in `flycheck-checkers` are command checkers, but command checkers are actually instances of *generic checkers*. Many external packages, such as `dafny-mode`, `fstar-mode`, etc. use generic checkers, which allow you more flexibility, including running Flycheck with persistent subprocess such as language servers. See flycheck-ocaml for an example of how to use a generic checker.

See also:

Asynchronous Processes(elisp) How to run and control asynchronous processes from inside Emacs.

Overlays(elisp) How to add temporary annotations to a buffer.

4.1.2 Adding a syntax checker to Flycheck

To add a syntax checker to Flycheck, you need to answer a few questions:

- How to invoke the checker? What is the name of its program, and what arguments should Flycheck pass to it?
- How to parse the error messages from the checker output?
- What language (or languages) will the checker be used for?

For instance, if I were to manually run the Scala compiler `scalac` on the following `hello.scala` file:

```
object {
  println("Hello, world")
}
```

Here is the output I would get:

```
$ scalac hello.scala
hello.scala:1: error: identifier expected but '{' found.
object {
       ^
one error found
```

The compiler reports one syntax error from the file `hello.scala`, on line 3, with severity `error`, and the rest of the line contains the error message.

So, if we want to instruct Flycheck to run `scalac` on our Scala files, we need to tell Flycheck to:

- Invoke `scalac FILE-NAME`
- Get errors from output lines of the form: `file-name:line:  error:message`

Writing the checker

Once you have answered these questions, you merely have to translate the answers to Emacs Lisp. Here is the full definition of the `scala` checker you can find in `flycheck.el`:

```
(flycheck-define-checker scala
  "A Scala syntax checker using the Scala compiler.

See URL `https://www.scala-lang.org/'."
  :command ("scalac" "-Ystop-after:parser" source)
```

(continues on next page)

(continued from previous page)

```
  :error-patterns
    ((error line-start (file-name) ":" line ": error: " (message) line-end))
  :modes scala-mode
  :next-checkers ((warning . scala-scalastyle)))
```

The code is rather self-explanatory; but we'll go through it nonetheless.

First, we define a checker using `flycheck-define-checker`. Its first argument, `scala`, is the name of the checker, as a symbol. The name is used to refer to the checker in the documentation, so it should usually be the name of the language to check, or the name of the program used to do the checking, or a combination of both. Here, `scalac` is the program, but the checker is named `scala`. There is another Scala checker using `scalastyle`, with the name `scala-scalastyle`. See *`flycheck-checkers`* for the full list of checker names defined in Flycheck.

After the name comes the docstring. This is a documentation string answering three questions: 1) What language is this checker for? 2) What is the program used? 3) Where can users get this program? Nothing more. In particular, this string does *not* include user documentation, which should rather go in the manual (see *Supported Languages*).

The rest of the arguments are keyword arguments; their order does not matter, but they are usually given in the fashion above.

- `:command` describes what command to run, and what arguments to pass. Here, we tell Flycheck to run `scalac -Ystop-after:parser` on `source`. In Flycheck, we usually want to get error feedback as fast as possible, hence we will pass any flag that will speed up the invocation of a compiler, even at the cost of missing out on some errors. Here, we are telling `scalac` to stop after the parsing phase to ensure we are getting syntax errors quickly.

  The `source` argument is special: it instructs Flycheck to create a temporary file containing the content of the current buffer, and to pass that temporary file as argument to `scalac`. That way, `scalac` can be run on the content of the buffer, even when the buffer has not been saved. There are other ways to pass the content of the buffer to the command, e.g., by piping it through standard input. These special arguments are described in the docstring of `flycheck-substitute-argument`.

- `:error-patterns` describes how to parse the output, using the `rx` regular expression syntax. Here, we expect `scalac` to return error messages of the form:

  ```
  file:line: error: message
  ```

  This is a common output format for compilers. With the following `:error-patterns` value:

  ```
  ((error line-start (file-name) ":" line ": error: " (message) line-end))
  ```

  we tell Flycheck to extract three parts from each line in the output that matches the pattern: the `file-name`, the `line` number, and the `message` content. These three parts are then used by Flycheck to create a *`flycheck-error`* with the `error` severity.

- `:modes` is the list of Emacs major modes in which this checker can run. Here, we want the checker to run only in `scala-mode` buffers.

That's it! This definition alone contains everything Flycheck needs to run `scalac` on a Scala buffer and parse its output in order to give error feedback to the user.

Note: `rx.el` is a built-in Emacs module for declarative regular expressions. Look for the documentation of the `rx` function inside Emacs for its usage. Flycheck extends `rx` with a few constructs like `line`, `file-name` and `message`. You can find them the full list in the docstring for `flycheck-rx-to-string`.

Registering the checker

Usually, you'll want to register the checker so that it is eligible for automatic selection. For that, you just need to add the checker symbol to *flycheck-checkers*. The order of checkers does matter, as only one checker can be enabled in a buffer at a time. Usually you want to put the most useful checker as the first checker for that mode. For instance, here are the JavaScript checkers provided by Flycheck:

```
javascript-eslint
javascript-jshint
javascript-gjslint
javascript-jscs
javascript-standard
```

If a buffer is in `js-mode`, Flycheck will try first to enable `javascript-eslint` before any other JavaScript checker.

There are other factors governing checker selection in a buffer, namely whether a checker is disabled by user configuration (see *Disable syntax checkers*), and whether this checker *can* be enabled (see the `:enabled` property in `flycheck-define-generic-checker`).

See also:

flycheck-get-checker-for-buffer This is the function that looks through *flycheck-checkers* to find a valid checker for the buffer.

Writing more complex checkers

Here are two examples of more complex checkers:

```
(flycheck-define-checker protobuf-protoc
  "A protobuf syntax checker using the protoc compiler.

See URL `https://developers.google.com/protocol-buffers/'."
  :command ("protoc" "--error_format" "gcc"
            (eval (concat "--java_out=" (flycheck-temp-dir-system)))
            ;; Add the file directory of protobuf path to resolve import directives
            (eval (concat "--proto_path=" (file-name-directory (buffer-file-name))))
            source-inplace)
  :error-patterns
  ((info line-start (file-name) ":" line ":" column
         ": note: " (message) line-end)
   (error line-start (file-name) ":" line ":" column
          ": " (message) line-end)
   (error line-start
          (message "In file included from") " " (file-name) ":" line ":"
          column ":" line-end))
  :modes protobuf-mode
  :predicate (lambda () (buffer-file-name)))
```

```
(flycheck-define-checker sh-shellcheck
  "A shell script syntax and style checker using Shellcheck.

See URL `https://github.com/koalaman/shellcheck/'."
  :command ("shellcheck"
            "--format" "checkstyle"
            "--shell" (eval (symbol-name sh-shell))
            (option-flag "--external-sources"
```

(continues on next page)

(continued from previous page)

```
                        flycheck-shellcheck-follow-sources)
            (option "--exclude" flycheck-shellcheck-excluded-warnings list
                    flycheck-option-comma-separated-list)
            "-")
  :standard-input t
  :modes sh-mode
  :error-parser flycheck-parse-checkstyle
  :error-filter (lambda (errors)
                  (flycheck-remove-error-file-names "-" errors))
  :predicate (lambda () (memq sh-shell '(bash ksh88 sh)))
  :verify
  (lambda (_)
    (let ((supported (memq sh-shell '(bash ksh88 sh))))
      (list (flycheck-verification-result-new
             :label (format "Shell %s supported" sh-shell)
             :message (if supported "yes" "no")
             :face (if supported 'success '(bold warning))))))
  :error-explainer
  (lambda (err)
    (let ((error-code (flycheck-error-id err))
          (url "https://github.com/koalaman/shellcheck/wiki/%S"))
      (and error-code `(url . ,(format url error-code))))))
```

The `:command` forms are longer, as the checkers pass more flags to `protoc` and `shellcheck`. Note the use of `eval`, `option`, and `option-flag` for transforming Flycheck checker options into flags for the command. See the docstring for `flycheck-substitute-argument` for more info, and look at other checkers for examples.

The `shellcheck` checker does no use `source` nor `source-inplace`: instead, it passes the buffer contents on standard input, using `:standard-input t`.

The `protoc` checker has three patterns in `:error-patterns`; the first one will catch `notes` from the compiler and turn them into *`flycheck-error`* objects with the `info` severity; the second is for errors from the file being checked, and the third one is for errors from other files. In the `shellcheck` checker, on the other hand, `:error-parser` replaces `:error-patterns`: `shellcheck` outputs results in the standard CheckStyle XML format, so the definition above uses Flycheck's built-in CheckStyle parser, and an `:error-filter` to replace `-` by the current buffer's filename.

Both checkers use a new `:predicate` property to determine when the checker can be called. In addition to the `:mode` property which restricts the `protoc` checker to buffers in `protobuf-mode`, the `:predicate` property ensures that `protoc` is called only when there is a file associated to the buffer (this is necessary since we are passing the file associated to the buffer `protobuf` using `source-inplace` in `:command`; in contrast, the `shellcheck` checker can run in all buffers, because it sends buffer contents through a pipe). The second checker has a more complex `:predicate` to make sure that the current shell dialect is supported, and a `:verify` function to help users diagnose configuration issues (`:verify` is helpful for giving feedback to users; its output gets included when users invoke *`flycheck-verify-setup`*)

Finally, the `shellcheck` checker includes an error explainer, which opens the relevant page on the ShellCheck wiki when users run *`flycheck-explain-error-at-point`*.

There are other useful properties, depending on your situation. Most important is `:enabled`, which is like `:predicate` but is run only once; it is used to make sure a checker has everything it needs before being allowed to run in a buffer (this is particularly useful when the checks are costly: running an external program and parsing its output, checking for a plugin, etc.).

See also:

flycheck-define-generic-checker For the full documentation of all the properties you can pass to `flycheck-define-checker`. Look also in the docstring for

`flycheck-define-command-checker` for additional properties.

Note: Don't be afraid to look into the `flycheck.el` code. The existing checkers serve as useful examples you can draw from, and all core functions are documented.

Sharing your checker

Once you have written your own syntax checker, why not submit a pull request to integrate it into Flycheck? If it's useful to you, it may be useful for someone else! Please do check out our *Contributor's Guide* to learn how we deal with pull requests.

Issues with auto-quoting in `flycheck-define-checker`

You may have noticed that lists passed to the `:command` or `:error-patterns` in the snippets above are not quoted. That is because `flycheck-define-checker` is a macro which automatically quotes these arguments (not unlike `use-package` and other configuration macros).

While this makes for less noisy syntax, it unfortunately prevents you from defining a checker with compile-time arguments. For example, you may be tempted to have a custom checker in your Emacs configuration written like this:

```
(flycheck-define-checker my-foobar-checker
  :command ("foobar" source)
  :error-patterns ((error ...))
  :modes `(foobar-mode ,my-other-foobar-mode))
```

The idea is that you know statically one mode that you want to use the checker in: `foobar-mode`, but another mode can be given via the variable `my-other-foobar-mode` before the checker is defined. This won't work, because the `:modes` property is auto-quoted by `flycheck-define-checker`. The issue arises not just with `:modes`:, but with almost all the other properties since they are also auto-quoted.

If you do find yourself in need to define such a checker, there is a solution though. The `flycheck-define-checker` macro is just a convenience over `flycheck-define-command-checker`, so you could define the checker above as follows:

```
(flycheck-def-executable-var my-foobar-checker "foobar")
(flycheck-define-command-checker 'my-foobar-checker
  :command '("foobar" source)
  :error-patterns '((error ...))
  :modes `(foobar-mode ,my-other-foobar-mode))
```

Using `flycheck-define-command-checker`, you now need to quote all the list arguments, but now with the confidence that no auto-quoting will take place, since `flycheck-define-command-checker` is just a function. Also note that you need to explicitly define the executable variable for the checker. Using `flycheck-define-command-checker` is the recommended way to define a checker with compile-time arguments.

Note: The `flycheck-define-checker` macro is an autoload, so using it inside a `with-eval-after-load` form will load all of Flycheck. While this ensures the macro is correctly expanded, it also defeats the purpose of using `with-eval-after-load`.

For the background behind this state of affairs, see issue 1398.

CHAPTER 5

The Contributor Guide

The Contributor Guide explains how to contribute to Flycheck.

5.1 Contributor's Guide

Thank you very much for your interest in contributing to Flycheck! We'd like to warmly welcome you in the Flycheck community, and hope that you enjoy your time with us!

There are many ways to contribute to Flycheck, and we appreciate all of them. We hope that this document helps you to contribute. If you have questions, please ask on our issue tracker or in our Gitter chatroom.

For a gentle start please take a look at all the things we need your help with and look for beginner-friendly tasks.

Please note that all contributors are expected to follow our *Code of Conduct*.

5.1.1 Bug reports

Bugs are a sad reality in software, but we strive to have as few as possible in Flycheck. Please liberally report any bugs you find. If you are not sure whether something is a bug or not, please report anyway.

If you have the chance and time please search existing issues, as it's possible that someone else already reported your issue. Of course, this doesn't always work, and sometimes it's very hard to know what to search for, so this is absolutely optional. We definitely don't mind duplicates, please report liberally.

To open an issue simply fill out the issue form. To help us fix the issue, include as much information as possible. When in doubt, better include too much than too little. Here's a list of facts that are important:

- What you did, and what you expected to happen instead
- Whether and how you were able to reproduce the issue in emacs -Q
- Your Flycheck setup from `M-x flycheck-verify-setup`

Windows-only issues

As Flycheck does not support Windows officially we generally do *not* attempt to fix issues that only occur on Windows. We will move all Windows-only issues to the list of open Windows issues, and leave them to Windows users and developers.

We welcome anyone who wants to fix open Windows issues, and we will merge pull requests for improved Windows compatibility. If you know Windows and Emacs, please take a look at the list of open Windows issues and try to fix any of these.

5.1.2 Feature requests

To request a new feature please open a new issue through our issue form. A feature request needs to find a core developer or maintainer who adopts and implements it.

5.1.3 The build system

Flycheck provides a `Makefile` with some convenient targets to compile and test Flycheck. The Makefile requires Cask, the Emacs Lisp dependency manager. Run `make help` to see a list of all available targets. Some common ones are:

- `make init` initialises the project by installing local Emacs Lisp dependencies.
- `make check` checks all Emacs Lisp sources. This target requires Emacs 25.
- `make compile` compiles Flycheck and its libraries to byte code.
- `make format` formats all Emacs Lisp sources.
- `make specs` runs all Buttercup specs for Flycheck. Set **`PATTERN`** to run only specs matching a specific regular expression, e.g. `make PATTERN='^Mode Line' specs` to run only tests for the mode line.
- `make unit` runs all ERT unit tests for Flycheck. We are phasing ERT out in favour of Buttercup; no new ERT unit tests will be added and this target will eventually be removed.
- `make integ` runs all integration tests for Flycheck syntax checkers. These tests are dependent on the checker programs and their versions; expect failures when running this target with bleeding-edge checkers. Set **`SELECTOR`** to run only tests matching a specific ERT selector, e.g. `make SELECTOR='(language haskell)' integ` to run only integration tests for Haskell. `make LANGUAGE=haskell integ` is a shortcut for this.

  If you want to replicate the integration tests that are run on the CI, continue reading.

5.1.4 Running all the integration tests

To run all the integration tests, you need to have all the syntax checkers installed. As that can be tedious work, and since your locally installed tools can have different versions than the tools used on the CI, we have created a Docker image with most of the supported checkers. To use the Docker image locally and replicate the integration tests that are run on the CI, first you need to build the image:

```
cd flycheck
docker pull flycheck/emacs-cask:26.2
docker pull flycheck/all-tools:latest
docker build --build-arg EMACS_VERSION=26.2 --tag tools-and-emacs:26.2 -f .travis/
↪tools-and-emacs .
```

Replace `26.2` by the Emacs version you want to test. See the available versions on docker hub.

Once the image is built, you can use it to run the integration tests:

```
docker run --rm -it -v `pwd`:/flycheck --workdir /flycheck tools-and-emacs:26.2 /bin/
↪bash -c "make integ"
```

Note that the `all-tools` image is rebuilt each month, so the versions of the its syntax checkers will change accordingly. You can check the version of each installed tool by running the `check-tools` script in the image:

```
docker run --rm -it -v `pwd`:/flycheck --workdir /flycheck tools-and-emacs:26.2 check-
↪tools
```

5.1.5 Pull requests

Pull Requests are the primary mechanism to submit your own changes to Flycheck. Github provides great documentation about Pull Requests.

Please make your pull requests against the `master` branch.

Use `make check specs unit` to test your pull request locally. When making changes to syntax checkers of a specific language, it's also a good idea to run `make LANGUAGE=`*`language`*` integ` and check whether the tests for the particular language still work. A successful `make integ` is by no means mandatory for pull requests, though, the continuous integration will test your changes, too.

Important: To contribute to Flycheck you must sign our CLA (Contributor License Agreement). The CLA Assistant bot will automatically ask you to do this when you open a pull request, and will let you sign the CLA through your Github account.

We require this process mostly to make you aware of the licensing implications of contributing to Flycheck and to obtain your explicit approval of our licenses for your contribution.

All pull requests go through a two-stage review process:

- *Maintainer* review the general idea and direction of the pull request and leave a `LGTM` comment if they believe that the change is a good addition to Flycheck. We currently require at least one approval from a maintainer.
- *All contributors*—language teams in particular—check the technical implementation of a pull request through pull request reviews, and either approve it or request changes. We currently require at least one approval and no requested changes.

Important: We have a comprehensive *Style Guide* that explains what features we will accept, how our code should look likewise, what tests we require, how commit messages should look like, and so on.

Take a look at it to see what we look for in a code review.

Additionally all pull requests go through automated tests on Travis CI which check code style, run unit tests, etc

Feel free to mention individual contributors or entire teams (e.g. `@flycheck/maintainers` or `@flycheck/javascript`) to ask for help or feedback or request a review. Please mention the maintainers (`@flycheck/maintainers`) if you think that your pull request has been waiting for review too long. You can expect a first response to any pull request in a couple of days.

Once the pull request passed review and automated tests we will merge it. We may also ask you whether you'd like to join Flycheck and help us, thus giving you commit access to our repository and let you merge your own pull request.

5.1.6 Writing documentation

Documentation improvements are very welcome. Flycheck's manual is written in reStructuredText and built with Sphinx. The source of the manual resides in the `doc/` directory.

You need Python 3.4 or newer to install Sphinx for Flycheck's documentation. On macOS it is recommended that you use Homebrew to install the latest Python version with `brew install python3`. On Linux you should be able to obtain Python 3.4 from the package manager of your distribution.

With Python 3 installed change into the `doc/` directory and run `make init` to install Sphinx and related tools required for Flycheck's documentation. We recommend that you use virtualenv to avoid a global installation of Python modules. `make init` will warn you if you do not.

When editing documentation run `make html-auto` to view the results of your edits. This target runs a local webserver at http://localhost:8000 which serves the HTML documentation and watches the documentation sources for changes to rebuild automatically. When you have finished your edits it is a good idea to run `make linkcheck` to verify all links in the documentation. Note that this target can take a while especially when run on a clean build.

Run `make help` to see a list of all available Make targets for the documentation.

Documentation pull requests work in the same way as other pull requests. To find documentation issues sort by the documentation label.

5.1.7 Issue management

We use Github labels for basic issue management:

- **The red "bug" label denotes critical bugs in Flycheck that must be fixed urgently.**
- Violet labels describe the area of Flycheck the issue belongs to.
- The green "beginner friendly" label denotes easy tasks for newcomers to the project.
- Orange labels denote blockers.
- Grey labels indicate resolutions to issues.

5.1.8 Out of tree contributions

There are many ways that you can contribute to Flycheck that go beyond this repository.

Answer questions in our Gitter channel or on StackExchange.

Participate in Flycheck discussions in other Emacs communities and help users with troubles.

Write *extensions for Flycheck*.

This contributing guide is heavily inspired by Rust's excellent contributing information.

5.2 Style Guide

This document describes our code style. It tells you what to look for when making changes to Flycheck, or when reviewing pull requests.

5.2.1 Features

Flycheck's scope and focus is providing the infrastructure and foundations for on-the-fly syntax checking. Flycheck provides the basics but deep integration with particular programming languages is best left to *separate packages*.

Whether a feature is within the scope of Flycheck is the *maintainer's* judgement call. Generally we reserve the right to reject any pull request for being out of scope.

- Avoid a *disproportionate amount of code* for a single syntax checker or language. Look at the built-in checkers for judgement. A syntax checker that requires a lot more code than any built-in checker is likely to be rejected.
- Avoid *deep integration* with a particular UI or completion framework. Emacs' standard is our standard: We will reject code that is tied to Helm or Counsel.
- Likewise do not deviate from Emacs' default behaviour too much. Stick to Emacs' standard for key bindings, interactive functions, etc.

5.2.2 Backward compatibility

Checkers and languages evolve over time, and their error format often change as a consequence. It is not a goal of Flycheck to work with every version of every checker ever supported. However, the latest Flycheck version *should always work* with the contemporary version of a checker.

As a rule of thumb, if maintaining backward compatibility is trivial (i.e., does not incur code maintenance costs), then we should do it. For example, a slightly more complex parsing regexp is OK, but doing version detection to add a flag would most likely be too much.

Keep in mind that users may not have the choice of updating to the latest version of a checker (e.g., `gcc` on Debian-based distributions). On the other hand, npm or Python packages are usually trivial to update. Making an extra effort to maintain backward compatibility for these hard-to-update checkers is reasonable.

The integration tests that are run on our CI should always reflect the latest supported version.

5.2.3 Style

Important: `make check compile` must pass on Emacs 25 or newer. This command checks for some formatting issues and compilation errors.

Run `make format` with Emacs 25 to automatically reformat the Emacs Lisp source files.

- Generally try to fit into the style of the code you see.
- Indent with the default indentation rules.
- Follow the *Programming Tips(elisp)* for Emacs Lisp.
- Whitespace:
  - 80 characters per line.
  - Avoid tabs and trailing spaces.
- Naming:
  - Prefix all variables and functions with the name of the containing library, i.e. `flycheck-` for everything that is in `flycheck.el`.
  - End boolean predicates with `-p`, i.e. `flycheck-valid-checker-p`.

- Avoid macros, and use them for syntax only.
- Adhere to the *Key Binding Conventions(elisp)*. Particularly do not define keys in Emacs' reserved keymaps or in the `C-c LETTER` space for user bindings.

5.2.4 Libraries

- Do **not** advise built-in or 3rd party functions and commands.
- Do **not** redefine built-in or 3rd party functions, unless for compatibility, but then copy the newer definition verbatim.
- Do **not** use `with-eval-after-load` and similar functions.
- Dependencies:
  - Use built-in Emacs libraries freely.
  - Introduce external dependencies with care. Prefer built-in libraries. `dash.el` is fine, though.
  - Avoid dependencies on language-specific libraries.
- Avoid `cl-lib`:
  - Prefer `seq` over `dash` over `cl-lib`. Use list functions from `cl-lib` only as the very last resort.
  - Prefer `let-alist` and `pcase` over `cl-destructuring-bind`.

5.2.5 Tests

- Add comprehensive buttercup specs for new functions and commands to `test/specs/`. Check whether the specs fit into an existing spec file, or add a new file instead. In doubt, use a new file.
- For new syntax checkers add at least one syntax checker integration test to `test/flycheck-test.el`. Make sure that the test passes with `make LANGUAGE=language integ`.

5.2.6 Documentation

- Add docstrings to all functions and variables.
- Follow the *Documentation Tips(elisp)*.
- Take care to update our manual:
  - Document new interactive commands and user options in the *user guide*.
  - Document new syntax checkers and new options for existing syntax checkers in the *list of languages*.
  - Document new or changed version requirements for syntax checkers in the *list of languages*.
  - Document changes to our build system and tooling in the *contributor's guide* or the *maintainer's guide*.

5.2.7 Commits

- Make each commit self-contained.
- Squash trivial fixes into previous commits so that no commit in and by itself violates this style guide.
- Write commit messages that adhere to the style illustrated below.

- In doubt prefer long messages over short messages. Take the time to write a good message that explains the intention of the change and illustrates noteworthy aspects of the implementation.
- If the commit fixes a bug try to reproduce a brief description of the bug in the message and make sure to mention the corresponding GitHub issue (e.g. `Fixes GH-42`).

Commit message style

This model commit message illustrates our style:

```
Fix a foo bug

The first line is the summary, 50 characters or less.  Write in the
imperative and in present tense: "Fix bug", not "fixed bug" or "fixes
bug".  Explain the intend of the change not the actual contents which the
diff already provides

After the summary more paragraphs with detailed explanations may follow,
wrapped at 72 characters.  Separate multiple paragraphs by blank lines.

You may use simple formatting like *emphasis* or _underline_, but keep
it to a minimum.  Commit messages are not in Markdown :)

Commit messages may reference issues by number, like this: See GH-42.
Please use `GH-` to prefix issue numbers.  You may also close issues
like this: Fixes GH-42 and closes GH-42.
```

Git Commit and Magit provide Emacs mode for Git commit messages, which helps you to comply to these guidelines.

See also:

A Note About Git Commit Messages Further information about good commit messages, including some motivation for our rules for commit messages.

5.3 Maintainer's Guide

5.3.1 Issue triage

Please label incoming tickets accordingly according to these rules:

- Add the "bug" label to things that you think **must be fixed urgently**. Please don't use this label for bugs that do not severely impede Flycheck's functionality.
- Add the "needs review" label to new bugs and pull requests that need to be reviewed.
- Add the "beginner friendly" label to really easy things. If you add this label please also add a comment that outlines a possible solution.
- Add "blocked" to bugs that need further comment or help from the reporter, and to pull requests that need to be improved.
- Add "needs help" to anything that no contributor will work on, to mark the issue as available for external contributors and inform users that we will not work on the issue.
- Add "windows only" for bugs that appear to only affect Windows operating systems.

If you'd like to review a bug or pull request please assign the corresponding ticket to you.

In issues for specific languages that Flycheck support please mention the corresponding *language team* if one exists.

5.3.2 Git workflow

Our Git workflow is simple:

- The `master` branch is always shippable.
- Every feature and every non-trivial change goes through a pull request.

GitHub calls this the "GitHub Flow" and has a very nice visual guide for this model.

Branch rules

Our workflow implies a couple of rules about which branches to push code to:

- Please commit new features, larger changes and refactorings and updates to documentation to separate branches and open a pull request for review and discussion.
- The `master` branch is protected. Only *owners* can push directly to it. Everyone else needs to open a pull request. Github requires maintainer approval and passing Travis CI tests before a pull request can be merged to master.

Important: When creating a new branch please use a *descriptive name* to communicate the purpose of the branch to other developers and maintainers. `fix-bug-42` is not a great name, but `42-fix-void-function-error-in-error-list` is.

Pull requests reviews

We review all pull requests, and require two different kinds of approval:

- At least one maintainer must approve the idea and direction with a `LGTM` comment.
- At least one contributor (maintainer or otherwise) must approve the implementation by leaving an approved pull request review, and no contributors must have requested changes.

As a maintainer

- Consider whether you personally think that the change is a good addition to Flycheck.
- Weight the expected benefits and impact of the feature against the expected complexity.
- Check whether the pull request complies with our *style guide*, but don't go too much into technical details.
- Don't review for technical details. It's the idea and direction that counts.

If you would like to see the pull request in Flycheck leave a `LGTM` comment.

As a contributor

- Check the technical implementation.
- Consider the impact on syntax checking for a language.
- Check whether the tests pass.
- Check whether the PR complies with our *style guide*.
- Challenge the technical implementation of a pull request, and ask questions about dubious code.
- Consider whether there might be a simpler approach or a better solution to the problem that the PR solves.

If you find any issues please leave a pull request review that requests for changes. Please try to leave an inline comment wherever possible and try to suggest a better solution, to make it easy for the PR author to discover and fix the issues.

If you didn't find any issues leave a pull request review that approves the changes.

In doubt request changes first and let the PR author explain their intention and implementation. You can still approve the review afterwards if you are satisfied.

Merge guidelines

Any contributor may merge approved pull requests. Our protection rules for the `master` branch ensure that only approved pull requests can be merged, but you still have to check a few things before merging:

- Are commits squashed? Before merging please take an extra look at the commits to make sure that the commits were properly squashed and have good commit messages. If needed, ask the contributor to improve the commit messages and squash the commits first, by requesting changes with a pull request review.
- Does the PR pass the integration tests? Not all checkers have integration tests, and not all tests are run on CI, so contributors should make sure to run them on their side.
- Should the PR warrant a line in the changelog? User-facing changes should be documented in `CHANGES.rst`.

For new features:

- Does the PR include tests? A new syntax checker should have at least one accompanying integration test.
- Does the PR include documentation? New syntax checkers or options should be documented in *Supported Languages*.

If all the points above have been addressed, then go ahead and click that green button :)

Note: We require proper merges for pull requests, to preserve the fact that a change came from a pull request in the git history and to retain any commit signatures that may exist. As such you can't squash-merge or rebase-merge through GitHub's UI.

Signatures for commits and tags

We sign all release tags as part of our *Release process*. Thus you need a GPG key pair for Git. Github provides a great guide which helps you to generate a key and to tell Git about your key. Please also add your key to your Github account.

We also recommend that you sign all your commits with your key. Again, Github provides a good guide to sign commits.

See also:

Signing Your Work For more information about signing commits and tags take a look at the section in the Git manual.

5.3.3 Tooling and Services

In addition to Github where we host code and do code reviews we use a bit of extra tooling and some 3rd party services for Flycheck:

- ReadTheDocs hosts http://www.flycheck.org and automatically rebuilds it on every change. It works mostly automatically and requires little configuration.
- Travis CI runs our tests after every push and for every pull request. It's configured through `.travis.yml`.
- CLA assistant checks signatures to our CLA and allows contributors to sign the CLA through their Github account.

All *maintainers* have administrative access to these services so in case of an issue just contact them.

5.3.4 Maintenance scripts

Administrative processes are tedious and time-consuming, so we try to automate as much as possible. The `maint/` directory contains many scripts for this purpose. `make -C maint/ help` provides an overview over all administrative tasks.

Most of these scripts require Python 3.5 and additional Python libraries. On OS X it is recommended that you use Homebrew to install the latest Python version with `brew install python3`. On Linux you should be able to obtain Python 3.5 from the package manager of your distribution.

To install all required libraries run `make -C maint init`. We recommend that you use virtualenv to avoid a global installation of Python modules. `make init` will warn you if you do not.

5.3.5 Versioning and releases

We use a single continuously increasing version number for Flycheck.

Important: Breaking changes may occur **at any point**.

Please feel free to make a release whenever you think it's appropriate. It's generally a good idea to release when

- you fixed an important bug that affects many users,
- there are a couple of new syntax checkers available,
- there's a major new feature in `master`,
- etc.

In doubt just make a release. We aim to release early and frequently. If anything breaks anything we can just publish another release afterwards.

Release process

First, check that

1. you are on `master`,
2. your working directory is clean, i.e. has no uncommitted changes or untracked files,

3. all commits are pushed,
4. and Travis CI passes for the latest commit on `master`.

If all is good a new release is a simple as

```
$ make -C maint release
```

This runs the release script in `maint/release.py`. If any of the above requirements isn't met the release script will signal an error and abort.

The release script bumps the version number, commits and tags a new release, and pushes it to Github.

Note: The tag is *signed*; you must configure Git for *signing commits and tags* before you make a release the first time. After pushing the new release to Github, the script bumps the version number again, to the next snapshot, and commits the changes again.

Once the script is completed please

1. Edit the release information on Github and add a short summary about the release. Don't forget to add a link to the complete changelog and upload the package TAR file.
2. Enable the new release on the ReadTheDocs versions dashboard.
3. Announce the new release in our Gitter channel, and wherever else you see fit.

5.3.6 New maintainers

To propose a new maintainer open a pull request that adds the user to `MAINTAINERS` and `doc/community/people.rst`. The pull request is subject to the *same rules* as all other pull requests. Notably it goes through the same approval process.

Once merged please also

- add the new maintainer to the `Maintainers` team of the Github organisation. This does not award additional privileges, it's just to support `@flycheck/maintainers` mentions for the sake of convenience,
- invite the new maintainer to the internal Maintainers channel on Gitter,

CHAPTER 6

Indices and Tables

- *Supported Languages*
- *Glossary*
- *Changes*
- genindex
- search

6.1 Supported Languages

This document lists all programming and markup languages which Flycheck supports.

Note: Extensions may provide support for additional languages or add deeper integration with existing languages.

Take a look at the *list of extensions* to see what the community can offer to you.

Each language has one or more syntax checkers whose names follow a convention of `language-tool`. All syntax checkers are listed in the order they would be applied to a buffer, with all available options. For more information about a syntax checker open Emacs and use **`flycheck-describe-checker`** to view the docstring of the syntax checker. Likewise, you may use **`describe-variable`** to read the complete docstring of any option.

6.1.1 Ada

`ada-gnat`
Check ADA syntax and types with GNAT.

`defcustom flycheck-gnat-args`
A list of additional options.

defcustom flycheck-gnat-include-path
A list of include directories. Relative paths are relative to the path of the buffer being checked.

defcustom flycheck-gnat-language-standard
The language standard to use as string.

defcustom flycheck-gnat-warnings
A list of additional warnings to enable. Each item is the name of a warning category to enable.

6.1.2 AsciiDoc

asciidoctor
Check AsciiDoc with the default Asciidoctor backend.

asciidoc
Check AsciiDoc with the standard AsciiDoc processor.

6.1.3 Awk

awk-gawk
Check Awk with gawk.

6.1.4 Bazel

bazel-buildifier
Check Bazel with buildifier.

6.1.5 C/C++

Flycheck checks C and C++ with either *c/c++-clang* or *c/c++-gcc*, and then with *c/c++-cppcheck*.

c/c++-clang
c/c++-gcc
Check C/C++ for syntax and type errors with Clang or GCC respectively.

> **Note:** *c/c++-gcc* requires GCC 4.4 or newer.

defcustom flycheck-clang-args
defcustom flycheck-gcc-args
A list of additional arguments for *c/c++-clang* and *c/c++-gcc* respectively.

defcustom flycheck-clang-blocks
Whether to enable blocks in *c/c++-clang*.

defcustom flycheck-clang-definitions
defcustom flycheck-gcc-definitions
A list of additional preprocessor definitions for *c/c++-clang* and *c/c++-gcc* respectively.

defcustom flycheck-clang-include-path
defcustom flycheck-gcc-include-path
A list of include directories for *c/c++-clang* and *c/c++-gcc* respectively, relative to the file being checked.

defcustom flycheck-clang-includes
defcustom flycheck-gcc-includes
A list of additional include files for *c/c++-clang* and *c/c++-gcc* respectively, relative to the file being checked.

defcustom flycheck-clang-language-standard
defcustom flycheck-gcc-language-standard
The language standard to use in *c/c++-clang* and *c/c++-gcc* respectively as string, via the `-std` option.

defcustom flycheck-clang-ms-extensions
Whether to enable Microsoft extensions to C/C++ in *c/c++-clang*.

defcustom flycheck-clang-no-exceptions
defcustom flycheck-gcc-no-exceptions
Whether to disable exceptions in *c/c++-clang* and *c/c++-gcc* respectively.

defcustom flycheck-clang-no-rtti
defcustom flycheck-gcc-no-rtti
Whether to disable RTTI in *c/c++-clang* and *c/c++-gcc* respectively, via `-fno-rtti`.

defcustom flycheck-clang-standard-library
The name of the standard library to use for *c/c++-clang*, as string.

defcustom flycheck-gcc-openmp
Whether to enable OpenMP in *c/c++-gcc*.

defcustom flycheck-clang-pedantic
defcustom flycheck-gcc-pedantic
Whether to warn about language extensions in *c/c++-clang* and *c/c++-gcc* respectively.

defcustom flycheck-clang-pedantic-errors
defcustom flycheck-gcc-pedantic-errors
Whether to error on language extensions in *c/c++-clang* and *c/c++-gcc* respectively.

defcustom flycheck-clang-warnings
defcustom flycheck-gcc-warnings
A list of additional warnings to enable in *c/c++-clang* and *c/c++-gcc* respectively. Each item is the name of a warning or warning category for `-W`.

c/c++-cppcheck
Check C/C++ for semantic and stylistic issues with cppcheck.

defcustom flycheck-cppcheck-checks
A list of enabled checks. Each item is the name of a check for the `--enable` option.

defcustom flycheck-cppcheck-inconclusive
Whether to enable inconclusive checks. These checks may yield more false positives than normal checks.

Note: This option requires cppcheck 1.54 or newer.

---

defcustom flycheck-cppcheck-include-path
A list of include directories. Relative paths are relative to the file being checked.

defcustom flycheck-cppcheck-standards
The C, C++ and/or POSIX standards to use via one or more `--std=` arguments.

defcustom flycheck-cppcheck-suppressions
The cppcheck suppressions list to use via one or more `--suppress=` arguments.

defcustom flycheck-cppcheck-suppressions-file
The cppcheck suppressions file to use via the `--suppressions-list=` argument.

6.1.6 CFEngine

cfengine
Check syntax with CFEngine.

6.1.7 Chef

chef-foodcritic
Check style in Chef recipes with foodcritic.

defcustom flycheck-foodcritic-tags
A list of tags to select.

6.1.8 Coffeescript

Flycheck checks Coffeescript syntax with *coffee* and then lints with *coffee-coffeelint*.

coffee
Check syntax with the Coffeescript compiler.

coffee-coffeelint
Lint with Coffeelint.

defcustom flycheck-coffeelintrc
Configuration file for this syntax checker. See *Configuration files*.

6.1.9 Coq

coq
Check and proof with the standard Coq compiler.

6.1.10 CSS

css-csslint
Check syntax and style with CSSLint.

css-stylelint
Syntax-check and lint CSS with stylelint.

defcustom flycheck-stylelintrc
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-stylelint-quiet
Whether to run stylelint in quiet mode via `--quiet`.

6.1.11 CUDA C/C++

cuda-nvcc
Checks syntax for CUDA C/C++ using the nvcc nvcc compiler bundled in the NVIDIA Toolkit.

CUDA C/C++ uses whichever system compiler you have configured, gcc/clang etc, but will sanitise error messages into a standardised format that can be picked up via flycheck. Corner cases may cause some odd behavior.

defcustom flycheck-cuda-language-standard
The C or C++ Language standard that you want the CUDA compiler to enforce.

defcustom flycheck-cuda-includes
A list of cuda includes.

defcustom flycheck-cuda-include-path
A list of include directories for nvcc.

defcustom flycheck-cuda-definitions
Additional preprocessor definitions for nvcc. Is passed unaltered to both GPU compiler and underlying C/C++ compiler.

6.1.12 CWL

cwl
Syntax check with (Schema Salad).

defcustom flycheck-cwl-schema-path
A path for the schema file for Common Workflow Language.

6.1.13 D

d-dmd
Check syntax and types with (DMD).

Note: This syntax checker requires DMD 2.066 or newer.

---

defcustom flycheck-dmd-include-path
A list of include directories.

defcustom flycheck-dmd-args
A list of additional arguments.

See also:

flycheck-d-unittest Flycheck extension which provides a syntax checker to run D unittests on the fly and report the results with Flycheck.

6.1.14 Dockerfile

dockerfile-hadolint
Check syntax and code style with hadolint

6.1.15 Elixir

elixir-credo
Check code style with credo

defcustom flycheck-elixir-credo-strict
When non-nil, run credo in strict mode, via `--strict`.

6.1.16 Emacs Lisp

Flycheck checks Emacs Lisp with *emacs-lisp* and then with *emacs-lisp-checkdoc*.

emacs-lisp
Check syntax with the built-in byte compiler.

defcustom flycheck-emacs-lisp-load-path
The load path as list of strings. Relative directories are expanded against the `default-directory` of the buffer being checked.

defcustom flycheck-emacs-lisp-initialize-packages
Whether to initialize Emacs' package manager with `package-initialize` before checking the buffer. If set to `auto` (the default), only initialize the package managers when checking files under `user-emacs-directory`.

defcustom flycheck-emacs-lisp-package-user-dir
The package directory as string. Has no effect if *flycheck-emacs-lisp-initialize-packages* is nil.

defcustom flycheck-emacs-lisp-check-declare
If non-nil, also check `declare-function` forms using `check-declare-file`.

emacs-lisp-checkdoc
Check Emacs Lisp documentation conventions with `checkdoc`.

See also:

Documentation Tips(elisp) Information about documentation conventions for Emacs Lisp.

purcell/flycheck-package Flycheck extension which adds a syntax checker to check for violation of Emacs Lisp library headers and packaging conventions.

Library Headers(elisp) Information about library headers for Emacs Lisp files.

6.1.17 Ember Templates

ember-template
Check your Ember templates with ember-template-lint

defcustom flycheck-ember-template-lintrc
Configuration file for this syntax checker. See *Configuration files*.

6.1.18 Erlang

Flycheck checks Erlang with *erlang-rebar3* in rebar projects and *erlang* otherwise.

erlang
Check Erlang with the standard Erlang compiler.

defcustom flycheck-erlang-include-path
A list of include directories.

defcustom flycheck-erlang-library-path
A list of library directories.

erlang-rebar3
Check Erlang with the rebar3 build tool.

defcustom flycheck-erlang-rebar3-profile
The profile to use when compiling, e.g. "default" or "test". The default value is nil which will use the test profile in test directories, the eqc profile in eqc directories and the default profile otherwise.

6.1.19 ERuby

eruby-erubis
Check ERuby with erubis.

eruby-ruumba
Check syntax and lint with Ruumba.

> **Note:** This syntax checker requires Ruumba 0.1.7 or newer.

defcustom flycheck-ruumba-lint-only
Whether to suppress warnings about style issues, via the `--lint` option.

defcustom flycheck-ruumbarc
Configuration file for this syntax checker. See *Configuration files*.

6.1.20 Fortran

fortran-gfortran
Check Fortran syntax and type with GFortran.

defcustom flycheck-gfortran-args
A list of additional arguments.

defcustom flycheck-gfortran-include-path
A list of include directories. Relative paths are relative to the file being checked.

defcustom flycheck-gfortran-language-standard
The language standard to use via the `-std` option.

defcustom flycheck-gfortran-layout
The source code layout to use. Set to `free` or `fixed` for free or fixed layout respectively, or nil (the default) to let GFortran automatically determine the layout.

defcustom flycheck-gfortran-warnings
A list of warnings enabled via the `-W` option.

6.1.21 Go

Flycheck checks Go with the following checkers:

1. *go-gofmt*

2. *go-golint*
3. *go-vet*
4. *go-build* or *go-test*
5. *go-errcheck*
6. *go-unconvert*
7. *go-staticcheck*

go-gofmt
Check Go syntax with gofmt.

go-golint
Check Go code style with Golint.

go-vet
Check Go for suspicious code with vet.

defcustom flycheck-go-vet-print-functions
A list of print-like functions to check calls for format string problems.

defcustom flycheck-go-build-tags
A list of build tags.

go-build
Check syntax and type with the Go compiler.

Note: This syntax checker requires Go 1.6 or newer.

---

defcustom flycheck-go-build-install-deps
Whether to install dependencies while checking with *go-build* or *go-test*

defcustom flycheck-go-build-tags
See *flycheck-go-build-tags*

go-test
Check syntax and types of Go tests with the Go compiler.

Note: This syntax checker requires Go 1.6 or newer.

---

defcustom flycheck-go-build-install-deps
See *flycheck-go-build-install-deps*.

defcustom flycheck-go-build-tags
See *flycheck-go-build-tags*

go-errcheck
Check for unhandled error returns in Go with errcheck.

Note: This syntax checker requires errcheck build from commit 8515d34 (Aug 28th, 2015) or newer.

---

defcustom flycheck-go-build-tags
See *flycheck-go-build-tags*

go-unconvert
Check for unnecessary type conversions with unconvert.

go-staticcheck
Perform static analysis and code linting with staticcheck, the successor to megacheck.

defcustom flycheck-go-version
staticcheck explicitly supports the last two releases of Go, but supports targeting older versions. Go versions should be specified like, "1.6", or, "1.11.4".

6.1.22 Groovy

groovy
Check syntax using the Groovy compiler.

6.1.23 Haml

haml
Check syntax with the Haml compiler.

6.1.24 Handlebars

handlebars
Check syntax with the Handlebars compiler.

6.1.25 Haskell

Flycheck checks Haskell with *haskell-stack-ghc* (in Stack projects) or *haskell-ghc*, and then with *haskell-hlint*.

See also:

flycheck-haskell Flycheck extension to configure Flycheck's Haskell checkers from the metadata, with support for Cabal sandboxes.

flycheck-hdevtools Flycheck extension which adds an alternative syntax checker for GHC using hdevtools.

haskell-stack-ghc
haskell-ghc
Check syntax and type GHC. In Stack projects invoke GHC through Stack to bring package dependencies from Stack in.

defcustom flycheck-ghc-args
A list of additional arguments.

defcustom flycheck-ghc-no-user-package-database
Whether to disable the user package database (only for *haskell-ghc*).

defcustom flycheck-ghc-stack-use-nix
Whether to enable Nix support for Stack (only for *haskell-stack-ghc*).

defcustom flycheck-ghc-stack-project-file
Allows to override the default `stack.yaml` file for Stack, via `--stack-yaml` (only for *haskell-stack-ghc*).

defcustom flycheck-ghc-package-databases
A list of additional package databases for GHC (only for *haskell-ghc*). Each item points to a directory containing a package directory, via `-package-db`.

defcustom flycheck-ghc-search-path
A list of module directories, via `-i`.

defcustom flycheck-ghc-language-extensions
A list of language extensions, via `-X`.

haskell-hlint
Lint with hlint.

defcustom flycheck-hlint-args
A list of additional arguments.

defcustom flycheck-hlint-language-extensions
A list of language extensions to enable.

defcustom flycheck-hlint-ignore-rules
A list of rules to ignore.

defcustom flycheck-hlint-hint-packages
A list of additional hint packages to include.

defcustom flycheck-hlintrc
Configuration file for this syntax checker. See *Configuration files*.

6.1.26 HTML

html-tidy
Check HTML syntax and style with Tidy HTML5.

defcustom flycheck-tidyrc
Configuration file for this syntax checker. See *Configuration files*.

6.1.27 Javascript

Flycheck checks Javascript with one of *javascript-eslint* or *javascript-jshint*.

Alternatively *javascript-standard* is used instead all of the former ones.

javascript-eslint
Check syntax and lint with ESLint.

Note: Flycheck automatically *disables* this syntax checker if eslint cannot find a valid configuration file for the current buffer.

---

defcustom flycheck-eslint-args
A list of additional arguments that are passed to eslint.

defcustom flycheck-eslint-rules-directories
A list of directories with custom rules.

javascript-jshint
Check syntax and lint with JSHint.

defcustom flycheck-jshint-extract-javascript
Whether to extract Javascript from HTML before linting.

defcustom flycheck-jshintrc
Configuration file for this syntax checker. See *Configuration files*.

javascript-standard
Check syntax and code style with Standard or Semistandard.

6.1.28 JSON

Flycheck checks JSON with *json-jsonlint*, *json-python-json*, or *json-jq*.

json-jsonlint
Check JSON with jsonlint.

json-python-json
Check JSON with Python's built-in `json` module.

json-jq
Check JSON with jq.

This checker accepts multiple consecutive JSON values in a single input, which is useful for jsonlines data.

6.1.29 Jsonnet

jsonnet
Checks Jsonnet with *jsonnet*.

6.1.30 Less

less
Check syntax with the Less compiler.

Note: This syntax checker requires lessc 1.4 or newer.

---

less-stylelint
Syntax-check and lint Less with stylelint.

defcustom flycheck-stylelintrc
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-stylelint-quiet
Whether to run stylelint in quiet mode via `--quiet`.

6.1.31 LLVM

llvm-llc
Check syntax with llc.

6.1.32 Lua

Flycheck checks Lua with *lua-luacheck*, falling back to *lua*.

lua-luacheck
Check syntax and lint with Luacheck.

defcustom flycheck-luacheckrc
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-luacheck-standards
The luacheck standards to use via one or more `--std` arguments.

lua
Check syntax with the Lua compiler.

6.1.33 Markdown

markdown-markdownlint-cli
Check Markdown with markdownlint-cli.

defcustom flycheck-markdown-markdownlint-cli-config
Configuration file for this syntax checker. See *Configuration files*.

markdown-mdl
Check Markdown with markdownlint.

defcustom flycheck-markdown-mdl-rules
A list of enabled rules.

defcustom flycheck-markdown-mdl-tags
A list of enabled rule tags.

defcustom flycheck-markdown-mdl-style
Configuration file for this syntax checker. See *Configuration files*.

6.1.34 Nix

nix
Check Nix with nix-instantiate.

nix-linter
Check Nix with nix-linter.

6.1.35 Opam

opam
Check Opam configuration files with opam lint.

6.1.36 Perl

Flycheck checks Perl with *perl* and *perl-perlcritic*.

perl
Check syntax with the Perl interpreter.

defcustom flycheck-perl-include-path
A list of include directories, relative to the file being checked.

defcustom flycheck-perl-module-list
A list of module names to implicitly use.

perl-perlcritic
Lint and check style with Perl::Critic.

defcustom flycheck-perlcritic-severity
The severity level as integer for the `--severity`.

defcustom flycheck-perlcritic-theme
The theme expression, passed as the `--theme` to `perlcritic`.

defcustom flycheck-perlcriticrc
Configuration file for this syntax checker. See *Configuration files*.

6.1.37 PHP

Flycheck checks PHP with *php*, *php-phpmd* and *php-phpcs*.

php
Check syntax with PHP CLI

php-phpmd
Lint with PHP Mess Detector.

defcustom flycheck-phpmd-rulesets
A list of rule sets. Each item is either the name of a default rule set, or the path to a custom rule set file.

php-phpcs
Check style with PHP Code Sniffer.

Note: This syntax checker requires PHP Code Sniffer 2.6 or newer.

---

defcustom flycheck-phpcs-standard
The coding standard, either as name of a built-in standard, or as path to a standard specification.

6.1.38 Processing

processing
Check syntax using the Processing compiler.

6.1.39 Protobuf

protobuf-protoc
Check syntax using the protoc compiler.

defcustom flycheck-protoc-import-path
A list of directories to resolve import directives. Relative paths are relative to the path of the buffer being checked.

protobuf-prototool
Lint with prototool.

6.1.40 Pug

pug
Check syntax using the Pug compiler.

6.1.41 Puppet

Flycheck checks Puppet with *puppet-parser* and lints with *puppet-lint*.

puppet-parser
Check syntax with the Puppet compiler.

puppet-lint
Link with Puppet Lint.

defcustom flycheck-puppet-lint-disabled-checks
A list of checks to disable.

defcustom flycheck-puppet-lint-rc
Configuration file for this syntax checker. See *Configuration files*.

6.1.42 Python

Flycheck checks Python with *python-flake8* or *python-pylint*, and falls back to *python-pycompile* if neither of those is available.

All Python checkers are invoked indirectly using `python -c ...` (rather than a direct call to `flake8` or `pylint`) to make it easier to switch between Python 2 and 3. For example, you can use `(setq flycheck-python-pylint-executable "python3")` to run `pylint` using Python 3, or `(defvaralias 'flycheck-python-flake8-executable 'python-shell-interpreter)` to run `flake8` through the executable pointed to by `python-shell-interpreter`.

Note: If Flycheck complains about a missing Python checker, make sure that the checker is reachable from `sys.path`, using e.g. `python -m pylint`: often, the issue is that the checker is installed globally but not in the current virtualenv. Alternatively, you can invoke the checker script directly, with `(setq flycheck-python-pylint-executable "pylint")`.

See also:

flycheck-pyflakes Flycheck extension which adds a syntax checker using Pyflakes.

msherry/flycheck-pycheckers Flycheck extension which can use multiple checkers simultaneously – including pyflakes, pep8, flake8, pylint, and mypy 2/3.

python-flake8
Check syntax and lint with flake8.

Note: This syntax checker requires flake8 3.0 or newer.

---

defcustom flycheck-flake8-error-level-alist
An alist mapping Flake8 error IDs to Flycheck error levels.

defcustom flycheck-flake8-maximum-complexity
The maximum McCabe complexity allowed for methods.

defcustom flycheck-flake8-maximum-line-length
The maximum length of lines.

defcustom flycheck-flake8rc
Configuration file for this syntax checker. See *Configuration files*.

python-pyright
Type check python with pyright.

Note: This syntax checker requires pyright.

python-mypy
Type check python with mypy.

Note: This syntax checker requires mypy 0.580 or newer.

defcustom flycheck-python-mypy-config
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-python-mypy-cache-dir
Directory used to write `.mypy_cache` directories.

Set to `null-device` to disable writing cache directories entirely.

python-pylint
Check syntax and lint with Pylint.

Note: This syntax checker requires Pylint 1.0 or newer.

defcustom flycheck-pylint-use-symbolic-id
Whether to report symbolic (e.g. `no-name-in-module`) or numeric (e.g. `E0611`) message identifiers.

defcustom flycheck-pylintrc
Configuration file for this syntax checker. See *Configuration files*.

python-pycompile
Check syntax with Python's byte compiler (see `py_compile`).

6.1.43 R

r-lintr
Check syntax and lint with lintr.

defcustom flycheck-lintr-caching
Whether to enable caching in lintr. On by default; it is not recommended to disable caching unless it causes actual problems.

defcustom flycheck-lintr-linters
Linters to use as a string with an R expression which selects the linters to use.

6.1.44 Racket

racket
Check syntax with raco expand from the `compiler-lib` package.

Note: This syntax checker needs the `compiler-lib` package.

6.1.45 RPM Spec

rpm-rpmlint
Lint with rpmlint.

6.1.46 reStructuredText

Flycheck checks reStructuredText with *rst-sphinx* in Sphinx projects and with *rst* otherwise.

rst-sphinx
Check documents with Sphinx.

> **Note:** This syntax checker requires Sphinx 1.2 or newer.

defcustom flycheck-sphinx-warn-on-missing-references
Whether to emit warnings for all missing references.

rst
Check documents with docutils.

6.1.47 Ruby

Flycheck checks Ruby with *ruby-rubocop*, *ruby-reek* and *ruby-rubylint*, falling back to *ruby* or *ruby-jruby* for basic syntax checking if those are not available.

ruby-rubocop
Check syntax and lint with RuboCop.

> **Note:** This syntax checker requires Rubocop 0.34 or newer.

defcustom flycheck-rubocop-lint-only
Whether to suppress warnings about style issues, via the `--lint` option.

defcustom flycheck-rubocoprc
Configuration file for this syntax checker. See *Configuration files*.

ruby-standard
Check syntax and lint with Ruby Standard.

> **Note:** This syntax checker and ruby-rubocop are mutually exclusive, since Standard employs an opinionated rubocop config.

defcustom flycheck-rubocop-lint-only
See *flycheck-rubocop-lint-only*.

defcustom flycheck-ruby-standardrc
Configuration file for this syntax checker. See *Configuration files*.

ruby-reek
Check syntax and lint with reek.

defcustom flycheck-reekrc
Configuration file for this syntax checker. See *Configuration files*.

Note: `flycheck-reekrc` defaults to `nil`, because Reek can find its own configuration.

ruby-rubylint
Check syntax and lint with ruby-lint.

Note: This syntax checker requires ruby-lint 2.0.2 or newer.

defcustom flycheck-rubylintrc
Configuration file for this syntax checker. See *Configuration files*.

ruby
Check syntax with the Ruby interpreter.

ruby-jruby
Check syntax with the JRuby interpreter.

6.1.48 Rust

Flycheck checks Rust with *rust-cargo* in Cargo projects, or *rust* otherwise. For Cargo projects, you can also use the clippy linter with *rust-clippy*.

rust-cargo
rust
rust-clippy
Check syntax and types with the Rust compiler. In a Cargo project the compiler is invoked through `cargo check` to take Cargo dependencies into account.

rust-clippy has no configurable options.

Note: *rust-cargo* requires Rust 1.17 or newer. *rust* requires Rust 1.18 or newer. *rust-clippy* requires the nightly version of Rust.

See also:

flycheck-rust Flycheck extension to configure Rust syntax checkers according to the current Cargo project.

defcustom flycheck-rust-args
A list of additional arguments that are passed to rustc. This option is ignored by *rust-cargo*.

defcustom flycheck-cargo-check-args
A list of additional arguments passed to the `cargo check` subcommand.

defcustom flycheck-rust-check-tests
Whether to check test code in Rust.

defcustom flycheck-rust-crate-root
A path to the crate root for the current buffer, or nil if the current buffer is a crate by itself.

rust-cargo ignores this option as the crate root is given by Cargo.

defcustom flycheck-rust-crate-type
For *rust-cargo*, the target type as a string, one of `lib`, `bin`, `example`, `test` or `bench`. Can also be nil for projects with a single target.

For *rust*, the type of the crate to check, as a string for the `--crate-type` option.

defcustom flycheck-rust-binary-name
The name of the binary to pass to `cargo check --TARGET-TYPE`, as a string.

For *rust-cargo*, always required unless *flycheck-rust-crate-type* is `lib` or nil, in which case it is ignored.

Ignored by *rust*.

defcustom flycheck-rust-features
List of features to activate during build or check.

The value of this variable is a list of strings denoting features that will be activated to build the target to check. Features will be passed to `cargo check --features=FEATURES`.

Empty by default.

Ignored by *rust*.

defcustom flycheck-rust-library-path
A list of additional library directories. Relative paths are relative to the buffer being checked.

6.1.49 Sass/SCSS

Flycheck checks SASS with *sass/scss-sass-lint*, falling back to *sass*, and SCSS with *scss-lint* or *scss-stylelint* falling back to *sass/scss-sass-lint* first and then *scss* if neither is available.

scss-lint
Syntax-check and lint SCSS with SCSS-Lint.

Note: This syntax checker requires SCSS-Lint 0.43.2 or newer.

defcustom flycheck-scss-lintrc
Configuration file for this syntax checker. See *Configuration files*.

sass/scss-sass-lint
Syntax-check and lint Sass/SCSS with SASS-Lint.

defcustom flycheck-sass-lintrc
Configuration file for this syntax checker. See *Configuration files*.

scss-stylelint
Syntax-check and lint SCSS with stylelint.

defcustom flycheck-stylelintrc
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-stylelint-quiet
Whether to run stylelint in quiet mode via `--quiet`.

sass
scss
Check SASS and SCSS respectively with the SCSS compiler.

defcustom flycheck-sass-compass
defcustom flycheck-scss-compass
Whether to enable the Compass CSS framework with `--compass`.

6.1.50 Scala

Flycheck checks Scala with *scala* and *scala-scalastyle*.

scala
Check syntax and types with the Scala compiler.

Note: This syntax checker is fairly primitive. For a better Scala experience we recommend Ensime.

scala-scalastyle
Check style with Scalastyle.

defcustom flycheck-scalastylerc
Configuration file for this syntax checker. See *Configuration files*.

Important: A configuration file is mandatory for this syntax checker. If *flycheck-scalastylerc* is not set or the configuration file not found this syntax checker will not be applied.

6.1.51 Scheme

Flycheck checks CHICKEN Scheme files with `csc`.

scheme-chicken

Check syntax with `csc`, the CHICKEN Scheme compiler.

defcustom flycheck-scheme-chicken-args

A list of additional options.

Important: Geiser must be installed and active for this checker to work.

6.1.52 Shell scripting languages

Flycheck checks various shell scripting languages:

- Bash with *sh-bash* and *sh-shellcheck*
- POSIX shell (i.e. `/bin/sh`) with *sh-posix-dash* or *sh-posix-bash*
- Zsh with *sh-zsh*

sh-bash
Check Bash syntax.

defcustom flycheck-sh-bash-args
A list of additional arguments that are passed to bash.

sh-posix-dash
Check POSIX shell syntax with Dash.

sh-posix-bash
Check POSIX shell syntax with Bash.

sh-zsh
Check Zsh syntax.

sh-shellcheck
Lint Bash and POSIX shell with ShellCheck.

defcustom flycheck-shellcheck-excluded-warnings
A list of excluded warnings.

defcustom flycheck-shellcheck-follow-sources
Allow shellcheck to read sourced files.

6.1.53 Slim

slim
Check Slim using the Slim compiler.

slim-lint
Check Slim best practices using the slim-lint linter.

6.1.54 SQL

sql-sqlint
Check SQL syntax with Sqlint.

6.1.55 systemd Unit Configuration

systemd-analyze
Check systemd unit configuration file syntax with systemd-analyze.

6.1.56 Tcl

tcl-nagelfar
Check Tcl syntax with Nagelfar.

6.1.57 Terraform

terraform
Check Terraform syntax with terraform fmt

terraform-tflint
Check Terraform with tflint

defcustom flycheck-tflint-variable-files
A list of files to resolve terraform variables. Relative paths are relative to the path of the buffer being checked.

6.1.58 Text

proselint
Check English prose with Proselint.

textlint
Check prose with textlint.

defcustom flycheck-textlint-config
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-textlint-plugin-alist
An alist mapping major modes to textlint plugins.

Flycheck currently supports the following textlint plugins on NPM:

- textlint-plugin-rst
- textlint-plugin-html
- textlint-plugin-latex
- textlint-plugin-asciidoctor (as well as other AsciiDoc plugins)

Note: textlint plugins need to be installed separately.

6.1.59 TeX/LaTeX

Flycheck checks TeX and LaTeX with either *tex-chktex* or *tex-lacheck*.

tex-chktex
Check style with ChkTeX.

defcustom flycheck-chktexrc
Configuration file for this syntax checker. See *Configuration files*.

tex-lacheck
Check style with Lacheck.

6.1.60 Texinfo

texinfo
Check syntax with **makeinfo** from Texinfo.

6.1.61 TypeScript

typescript-tslint
Check syntax and style with TSLint.

defcustom flycheck-typescript-tslint-config
Configuration file for this syntax checker. See *Configuration files*.

defcustom flycheck-typescript-tslint-rulesdir
Additional rules directory, for user created rules.

defcustom flycheck-tslint-args
A list of additional arguments that are passed to tslint.

6.1.62 Verilog

verilog-verilator
Check syntax with Verilator.

defcustom flycheck-verilator-include-path
A list of include directories. Relative paths are relative to the file being checked.

6.1.63 VHDL

vhdl-ghdl
Check syntax with GHDL.

defcustom flycheck-ghdl-language-standard
The language standard to use as string.

defcustom flycheck-ghdl-workdir
The directory to use for the file library.

defcustom flycheck-ghdl-ieee-library
The standard to use for the IEEE library.

6.1.64 XML

Flycheck checks XML with *xml-xmlstarlet* or *xml-xmllint*.

xml-xmlstarlet
Check syntax with XMLStarlet.

defcustom flycheck-xml-xmlstarlet-xsd-path
defcustom flycheck-xml-xmllint-xsd-path
Location of XSD schema to validate against for *xml-xmlstarlet* and *xml-xmllint* respectively.

xml-xmllint
Check syntax with **xmllint** from Libxml2.

6.1.65 YAML

Flycheck checks YAML with *yaml-jsyaml*, *yaml-ruby* or 'yaml-yamllint'.

yaml-jsyaml
Check syntax with js-yaml.

yaml-ruby
Check syntax with Ruby's YAML parser.

yaml-yamllint
Check syntax with yamllint.

defcustom flycheck-yamllintrc
Configuration file for this syntax checker. See *Configuration files*.

6.2 Glossary

The glossary explains most of the special terms we use in this documentation. some of these are originally explained in the Emacs manual or the Emacs Lisp reference, but we reproduce them here for convenience.

init file

user init file Your main Emacs configuration file. It's typically located in your *user emacs directory* at `$HOME/.emacs.d/init.el`. Emacs also looks at `$HOME/.emacs`, but this location is not recommended anymore. To find out the actual path to your init file of your Emacs session inspect the value of the variable `user-init-file` with `C-h v user-init-file`. You can visit it directly with `M-: (find-file user-init-file)`.

See also:

Init File(emacs) More information about the init file.

Init File(elisp) Programming interface for the init file.

user emacs directory The directory for all Emacs related files of the current user, at `~/.emacs.d/`. Many Emacs packages create data files in this directory, and it holds the recommended location for the *init file* at `~/.emacs.d/init.el`.

registered syntax checker A syntax checker in *`flycheck-checkers`*. Flycheck will only use these syntax checkers when checking buffers automatically.

verification buffer A buffer shown by `M-x flycheck-verify-setup`. This buffer contains information about the Flycheck setup for the current buffer.

executable option

executable options Options to override the executables of syntax checkers that run external commands. They are named `flycheck-`*`checker`*`-executable`, e.g. `flycheck-c/c++-clang-executable` for *`c/c++-clang`*.

Flycheck implicit defines these options for all syntax checkers defined with `flycheck-define-checker`.

6.3 Changes

6.3.1 33-cvs (in development)

- New features and improvements
  - The `flycheck-verify-setup` UI now includes buttons to re-enable manually disabled checkers and to try to re-enable automatically disabled checkers (command checkers are automatically disabled when their executable cannot be found). [GH-1755]
  - Error explainers can now return URLs (to show a webpage) or functions (to use custom formatting). For example, the Rust checker now renders explanations using `markdown-view-mode`. [GH-1753]
- **Breaking changes**
  - The variable `flycheck-current-errors` now contains errors in the order in which they were returned by checkers. In previous versions of Flycheck, this list was sorted by error position and severity. [GH-1749]

6.3.2 32-cvs (frozen on May 3rd, 2020)

- Highlights
  - Many checkers and compiler, such as `ocaml`, `rust`, `eslint`, and others, include end-line and end-column information. Flycheck can now highlight the exact region that they report. Authors of checker definitions can use the new `:end-line` and `:end-column` arguments in `flycheck-error-new`, or the new `end-line` and `end-column` fields in error patterns. [GH-1400]
  - Errors that checkers return for other files will now be displayed on the first line of the current buffer instead of begin discarded. The error list indicates which file each error came from, and navigation moves automatically moves between files. This change helps with compiled languages, where an error in another file may cause the current file to be considered invalid. Variables `flycheck-relevant-error-other-file-show` and `flycheck-relevant-error-other-file-minimum-level` control this behavior. [GH-1427]
  - Flycheck can now draw error indicators in margins in addition to fringes. Margins can contain arbitrary characters and images, not just monochrome bitmaps, allowing for a better experience on high-DPI screens. `flycheck-indication-mode` controls this behavior, and `flycheck-set-indication-mode` can be used to automatically adjust the fringes and margins. Additionally, Flycheck's will now use high-resolution fringe bitmaps if the fringe is wide enough [GH-1742, GH-1744]
  - Error highlighting is now configurable, using the new `flycheck-highlighting-style` variable: instead of applying level-dependent faces (typically with wavy underlines), Flycheck can now insert delimiters around errors, or mix styles depending on how many lines an error covers. Additionally, stipples are added in the fringes to indicate errors that span multiple lines. [GH-1743]
- New features and improvements
  - Flycheck can now trigger a syntax check automatically after switching buffers, using the `idle-buffer-switch` option in `flycheck-check-syntax-automatically`. This is useful when errors in a file are due to problems in a separate file. Variables `flycheck-idle-buffer-switch-delay` and `flycheck-buffer-switch-check-intermediate-buffers` control the functionality. [GH-1297]
  - Flycheck will now use Emacs' native XML parsing when libXML fails. This behavior can be changed by customizing `flycheck-xml-parser`. [GH-1349]
  - `flycheck-verify-setup` now shows more clearly which checkers will run in the buffer, and which are misconfigured. [GH-1478]
  - Flycheck now locates checker executables using a customizable function, `flycheck-executable-find`. The default value of this function allows relative paths (set e.g. in file or dir-local variables) in addition to absolute paths and executable names. [GH-1485]
  - Checkers that report error positions as a single offset from the start of the file can use the new `flycheck-error-new-at-pos` constructor instead of converting that position to a line and a column. [GH-1400]
  - Config-file variables can now be set to a list of file names. This is useful for checkers like mypy which don't run correctly when called from a subdirectory without passing an explicit config file. [GH-1711]
  - Thanks to algorithmic improvements in error reporting, Flycheck is now much faster in large buffers. [GH-1750]
- New syntax checkers:
  - Awk with `gawk` [GH-1708]
  - Bazel with `bazel-buildifier` [GH-1613]

  - CUDA with `cuda-nvcc` [GH-1508]
  - CWL with `schema-salad-tool` [GH-1361]
  - Elixir with `credo` [GH-1062]
  - JSON with `json-jq` [GH-1568]
  - Jsonnet with `jsonnet` [GH-1345]
  - MarkdownLint CLI with `markdownlint` [GH-1366]
  - mypy with `python-mypy` [GH-1354]
  - Nix with `nix-linter` [GH-1530]
  - Opam with `opam lint` [GH-1532]
  - protobuf-prototool with `prototool` [GH-1591]
  - Rust with `rust-clippy` [GH-1385]
  - Ruumba with `eruby-ruumba` [GH-1616]
  - Staticcheck with `go-staticheck` [GH-1541]
  - terraform with `terraform fmt`, `tflint` [GH-1586]
  - Tcl with `nagelfar` [GH-1365]
  - Text prose with `textlint` [GH-1534]
  - VHDL with `ghdl` [GH-1160]
- Checker improvements:
  - `python-pylint` and `python-flake8` are now invoked with `python -c`, to make it easier to change between Python 2 and Python 3. [GH-1113]
  - Add `flycheck-perl-module-list` to use specified modules when syntax checking code with the `perl` checker. [GH-1207]
  - `rust-cargo` now uses `cargo check` and `cargo test`. [GH-1289]
  - Add `flycheck-ghc-stack-project-file` for the `haskell-stack-ghc` checker. [GH-1316]
  - Add `flycheck-cppcheck-suppressions-file` to pass a suppressions file to cppcheck. [GH-1329]
  - Add `--force-exclusion` flag to `rubocop` command. [GH-1348]
  - Flycheck now uses ESLint's JSON output instead of checkstyle XML. [GH-1350]
  - Add `flychjeck-eslint-args` to pass arguments to `javascript-eslint`. [GH-1360]
  - Flycheck will now execute `rubocop` from the directory where a `Gemfile` is located. If a `Gemfile` does not exist, the old behaviour of running the command from the directory where `.rubocop.yml` is found will be used. [GH-1368]
  - Add `flycheck-sh-bash-args` to pass arguments to `sh-bash`. [GH-1439]
  - `haskell-stack-ghc` will not try to install GHC anymore. [GH-1443]
  - Add `flycheck-ghdl-ieee-library` to select which standard IEEE library to use for ghdl. [GH-1547]
  - The `javascript-eslint` checker now supports `typescript-mode` by default.

  - Add `flycheck-erlang-rebar3-profile` to select which profile to use when compiling erlang with rebar3. [GH-1560]
  - Add `flycheck-relevant-error-other-file-show` to avoid showing errors from other files. [GH-1579]
  - The `nix-linter` checker now has an error explainer. [GH-1586]
  - The Emacs Lisp checker can now run in buffers not backed by files. [GH-1695]
- **Breaking changes**
  - Remove the `javascript-jscs` checker. [GH-1024]
  - Remove the `elixir-dogma` checker. [GH-1450]
  - `rust-cargo` now requires Rust 1.17 or newer. [GH-1289]
  - `rust` now requires 1.18 or newer. [GH-1501]
  - Rename `flycheck-cargo-rustc-args` to `flycheck-cargo-check-args`. [GH-1289]
  - `rust-cargo` does not use the variable `flycheck-rust-args` anymore. [GH-1289]
  - Improve detection of default directory for `haskell-ghc` to consider `hpack` project files. [GH-1435]
  - Replace `go tool vet` with `go vet`. [GH-1548]
  - Remove the deprecated `go-megacheck` checker, which is replaced by `go-staticcheck`. [GH-1583]

6.3.3 31 (Oct 07, 2017)

- **Breaking changes**
  - `rust-cargo` now requires Rust 1.15 or newer [GH-1201]
  - Remove javascript-gjslint checker
- New syntax checkers:
  - Protobuf with `protoc` [GH-1125]
  - systemd-analyze with `systemd-analyze` [GH-1135]
  - Nix with `nix-instantiate` [GH-1164]
  - Dockerfile with `hadolint` [GH-1194]
  - AsciiDoc with `asciidoctor` [GH-1167]
  - CSS/SCSS/LESS with `stylelint` [GH-903]
  - Ruby with `reek` [GH-1244]
  - Go with `megacheck` [GH-1290]
  - LLVM IR with `llc` [GH-1302]
  - Text prose with `proselint` [GH-1304]
- New features:
  - Add `flycheck-xml-xmlstarlet-xsd-path` and `flycheck-xml-xmllint-xsd-path` to specify an XSD schema to validate XML documents against [GH-1272]
  - Add `flycheck-tslint-args` to pass additional arguments to tslint [GH-1186]
  - Add an error explainer to the `rpm-rpmlint` checker using `rpmlint -I` [GH-1235]

  - Add `flycheck-emacs-lisp-check-declare` to check function declaration in the `emacs-lisp` checker [GH-1286]
  - Add `flycheck-shellcheck-follow-sources` to check included files when using the `sh-shellcheck` checker [GH-1256]
- Improvements:
  - Use option `flycheck-go-build-tags` for `go-test`, `go-vet` and `go-errcheck` as well.
  - Add a revert function to `flycheck-verify-setup`, so hitting `g` reloads the buffer.
  - Make sure the erlang compiler is only run on compilable files.
  - `flycheck-tslint` does not crash any more on deprecation notices [GH-1174]
  - `rust-cargo` now checks integration tests, examples and benchmarks [GH-1206]
  - `rust-cargo` does not use `flycheck-rust-library-path` anymore, as dependencies are taken care of by Cargo [GH-1206]
  - `c/c++-gcc` checker now works from GCC 4.4 and up [GH-1226]

6.3.4 30 (Oct 12, 2016)

- **Breaking changes**
  - Flycheck now requires flake8 3.0 or newer
  - Remove `--config` option in `lua-luacheck` in favour of `luacheck`'s own `.luacheckrc` detection. Therefore `flycheck-luacheckrc` is no longer used [GH-1057]
  - `:modes` is now mandatory for syntax checker definitions [GH-1071]
  - Remove jade checker [GH-951] [GH-1084]
  - Remove `javascript-eslintrc` and instead rely on eslint's own configuration file search [GH-1085]
  - `C-c ! e` explains errors now [GH-1122]
- New syntax checkers:
  - Elixir with `dogma` [GH-969]
  - sass and scss with `sass-lint` [GH-1070]
  - Pug [GH-951] [GH-1084]
- New features:
  - Add `flycheck-cargo-rustc-args` to pass multiple arguments to cargo rustc subcommand [GH-1079]
  - Add `:error-explainer` to `flycheck-define-checker` and `flycheck-explain-error-at-point` to display explanations of errors [GH-1122]
  - Add an error explainer to the `rust` and `rust-cargo` checkers using `rustc --explain` [GH-1122]
  - Add `:enabled` property to `flycheck-define-checker` [GH-1089]
- Improvements:
  - Do not use `javascript-eslint` if eslint cannot find a valid configuration [GH-1085]
  - Automatically disable syntax checkers which are not installed instead of checking executable before each syntax check [GH-1116]

  - Add patterns for syntax errors to `scheme-chicken` [GH-1123]

6.3.5 29 (Aug 28, 2016)

- **Breaking changes**
  - Change `flycheck-eslint-rulesdir` (string) to `flycheck-eslint-rules-directories` (list of strings) [GH-1016]
  - Require rust 1.7 or newer for `rust` and `rust-cargo` [GH-1036]
- New syntax checkers:
  - Slim with `slim-lint` [GH-1013]
  - CHICKEN Scheme with `csc` [GH-987]
- New features:
  - Add `:working-directory` option to `flycheck-define-command-checker` [GH-973] [GH-1012]
  - `flycheck-go-build-install-deps` turns on dependency installation for `go test` as well as `go build` [GH-1003]
- Improvements:
  - Add default directory for `haskell-stack-ghc` and `haskell-ghc` checkers [GH-1007]
  - `rust` and `rust-cargo` checkers now support the new error format of rust 1.12 [GH-1016]
  - `flycheck-verify-checker` and `flycheck-verify-setup` now include information about configuration files of syntax checkers [GH-1021] [GH-1038]

6.3.6 28 (Jun 05, 2016)

- **Breaking changes**:
  - Rename `luacheck` to `lua-luacheck` to comply with our naming conventions
  - Remove `flycheck-cppcheck-language-standard` in favour of `flycheck-cppcheck-standards` which is a list of standards [GH-960]
- New features:
  - Add option to set binary name for `rust-cargo` [GH-958]
  - Add `flycheck-cppcheck-standards` to pass multiple code standards to cppcheck [GH-960]
  - Add `flycheck-cppcheck-suppressions` to suppress warnings for cppcheck [GH-960]
- Improvements:
  - Check Racket syntax in Geiser Mode [GH-979]
- Bug fixes
  - Do not signal errors when tslint reports no output [GH-981]
  - Do not generate invalid temporary filenames on Windows [GH-983]

6.3.7 27 (May 08, 2016)

- **Breaking changes**
  - Require PHP Code Sniffer 2.6 or newer for `php-phpcs` [GH-921]
- New syntax checkers:
  - Go with `go-unconvert` [GH-905]
  - Markdown with `mdl` [GH-839] [GH-916]
  - TypeScript with `tslint` [GH-947] [GH-949]
- Improvements:
  - Pass checkdoc settings from Emacs to *`emacs-lisp-checkdoc`* [GH-741] [GH-937]
- Bug fixes:
  - Fix parsing of syntax errors in triple-quoted strings for `python-pycompile` [GH-948]
  - Correctly handle rules based on the current file name in `php-phpcs` [GH-921]

6.3.8 26 (Apr 27, 2016)

Flycheck now has a Code of Conduct which defines the acceptable behaviour and the moderation guidelines for the Flycheck community. [GH-819]

Flycheck also provides a Gitter channel now for questions and discussions about development. [GH-820]

The native Texinfo manual is again replaced with a Sphinx based documentation. We hope that this change makes the manual easier to edit and to maintain and more welcoming for new contributors. The downside is that we can not longer include a Info manual in Flycheck's MELPA packages.

From this release onward Flycheck will use a single continuously increasing version number. Breaking changes may occur at any point.

- **Breaking changes**:
  - Remove `flycheck-copy-messages-as-kill`, obsolete since Flycheck 0.22
  - Remove `flycheck-perlcritic-verbosity`, obsolete since Flycheck 0.22
  - Replace `flycheck-completion-system` with `flycheck-completing-read-function` [GH-870]
  - JSON syntax checkers now require `json-mode` and do not check in Javascript Mode anymore
  - Prefer eslint over jshint for Javascript
  - Obsolete `flycheck-info` in favour of the new `flycheck-manual` command
- New syntax checkers:
  - Processing [GH-793] [GH-812]
  - Racket [GH-799] [GH-873]
- New features:
  - Add `flycheck-puppet-lint-rc` to customise the location of the puppetlint configuration file [GH-846]
  - Add `flycheck-puppet-lint-disabled-checks` to disable specific checks of puppetlint [GH-824]

  - New library `flycheck-buttercup` to support writing Buttercup specs for Flycheck
  - Add `flycheck-perlcriticrc` to set a configuration file for Perl::Critic [GH-851]
  - Add `flycheck-jshint-extract-javascript` to extract Javascript from HTML [GH-825]
  - Add `flycheck-cppcheck-language-standard` to set the language standard for cppcheck [GH-862]
  - Add `flycheck-mode-line-prefix` to customise the prefix of Flycheck's mode line lighter [GH-879] [GH-880]
  - Add `flycheck-go-vet-shadow` to check for shadowed variables with `go vet` [GH-765] [GH-897]
  - Add `flycheck-ghc-stack-use-nix` to enable Nix support for Stack GHC [GH-913]
- Improvements:
  - Map error IDs from flake8-pep257 to Flycheck error levels
  - Explicitly display errors at point with `C-c ! h` [GH-834]
  - Merge message and checker columns in the error list to remove redundant ellipsis [GH-828]
  - Indicate disabled checkers in verification buffers [GH-749]
  - Do not enable Flycheck Mode in `fundamental-mode` buffers [GH-883]
  - Write `go test` output to a temporary files [GH-887]
  - Check whether `lintr` is actually installed [GH-911]
- Bug fixes:
  - Fix folding of C/C++ errors from included files [GH-783]
  - Fix verification of SCSS-Lint checkstyle reporter
  - Don't fall back to `rust` if `rust-cargo` should be used [GH-817]
  - Don't change current buffer when closing the error message buffer [GH-648]
  - Never display error message buffer in current window [GH-822]
  - Work around a caching issue in Rubocop [GH-844]
  - Fix checkdoc failure with some Emacs Lisp syntax [GH-833] [GH-845] [GH-898]
  - Correctly parse Haskell module name with exports right after the module name [GH-848]
  - Don't hang when sending buffers to node.js processes on Windows [GH-794][GH-850]
  - Parse suggestions from `hlint` [GH-874]
  - Go errcheck handles multiple `$GOPATH` entries correctly now [GH-580][GH-906]
  - Properly handle Go build failing in a directory with multiple packages [GH-676] [GH-904]
  - Make cppcheck recognise C++ header files [GH-909]
  - Don't run phpcs on empty buffers [GH-907]

CHAPTER 7

Licensing

Flycheck 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.

Flycheck 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.

See *GNU General Public License 3* for a copy of the GNU General Public License.

You may copy, distribute and/or modify the Flycheck documentation under the terms of the Creative Commons Attribution-ShareAlike 4.0 International Public License. See *Creative Commons Attribution-ShareAlike 4.0 International* for a copy of the license.

Permission is granted to copy, distribute and/or modify the Flycheck logo under the terms of the Creative Commons Attribution-ShareAlike 4.0 International Public License. See *Creative Commons Attribution-ShareAlike 4.0 International* for a copy of the license.

7.1 Flycheck licenses

7.1.1 GNU General Public License 3

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.
```

(continues on next page)

(continued from previous page)

```
  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
```

(continues on next page)

(continued from previous page)

```
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
```

(continues on next page)

(continued from previous page)

```
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
```

(continues on next page)

(continued from previous page)

```
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
```

(continues on next page)

(continued from previous page)

```
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.
```

(continues on next page)

(continued from previous page)

```
  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.
```

(continues on next page)

(continued from previous page)

```
  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
```

(continues on next page)

(continued from previous page)

```
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
```

(continues on next page)

(continued from previous page)

```
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
```

(continues on next page)

(continued from previous page)

```
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
```

(continues on next page)

(continued from previous page)

```
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
```

(continues on next page)

(continued from previous page)

```
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program 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 this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

7.1.2 Creative Commons Attribution-ShareAlike 4.0 International

```
Attribution-ShareAlike 4.0 International

=======================================================================

Creative Commons Corporation ("Creative Commons") is not a law firm and
does not provide legal services or legal advice. Distribution of
Creative Commons public licenses does not create a lawyer-client or
other relationship. Creative Commons makes its licenses and related
information available on an "as-is" basis. Creative Commons gives no
warranties regarding its licenses, any material licensed under their
terms and conditions, or any related information. Creative Commons
disclaims all liability for damages resulting from their use to the
fullest extent possible.

Using Creative Commons Public Licenses
```

(continues on next page)

(continued from previous page)

```
Creative Commons public licenses provide a standard set of terms and
conditions that creators and other rights holders may use to share
original works of authorship and other material subject to copyright
and certain other rights specified in the public license below. The
following considerations are for informational purposes only, are not
exhaustive, and do not form part of our licenses.

     Considerations for licensors: Our public licenses are
     intended for use by those authorized to give the public
     permission to use material in ways otherwise restricted by
     copyright and certain other rights. Our licenses are
     irrevocable. Licensors should read and understand the terms
     and conditions of the license they choose before applying it.
     Licensors should also secure all rights necessary before
     applying our licenses so that the public can reuse the
     material as expected. Licensors should clearly mark any
     material not subject to the license. This includes other CC-
     licensed material, or material used under an exception or
     limitation to copyright. More considerations for licensors:
        wiki.creativecommons.org/Considerations_for_licensors

     Considerations for the public: By using one of our public
     licenses, a licensor grants the public permission to use the
     licensed material under specified terms and conditions. If
     the licensor's permission is not necessary for any reason--for
     example, because of any applicable exception or limitation to
     copyright--then that use is not regulated by the license. Our
     licenses grant only permissions under copyright and certain
     other rights that a licensor has authority to grant. Use of
     the licensed material may still be restricted for other
     reasons, including because others have copyright or other
     rights in the material. A licensor may make special requests,
     such as asking that all changes be marked or described.
     Although not required by our licenses, you are encouraged to
     respect those requests where reasonable. More_considerations
     for the public:
        wiki.creativecommons.org/Considerations_for_licensees

=======================================================================

Creative Commons Attribution-ShareAlike 4.0 International Public
License

By exercising the Licensed Rights (defined below), You accept and agree
to be bound by the terms and conditions of this Creative Commons
Attribution-ShareAlike 4.0 International Public License ("Public
License"). To the extent this Public License may be interpreted as a
contract, You are granted the Licensed Rights in consideration of Your
acceptance of these terms and conditions, and the Licensor grants You
such rights in consideration of benefits the Licensor receives from
making the Licensed Material available under these terms and
conditions.


Section 1 -- Definitions.

  a. Adapted Material means material subject to Copyright and Similar
```

(continues on next page)

(continued from previous page)

```
   Rights that is derived from or based upon the Licensed Material
   and in which the Licensed Material is translated, altered,
   arranged, transformed, or otherwise modified in a manner requiring
   permission under the Copyright and Similar Rights held by the
   Licensor. For purposes of this Public License, where the Licensed
   Material is a musical work, performance, or sound recording,
   Adapted Material is always produced where the Licensed Material is
   synched in timed relation with a moving image.

b. Adapter's License means the license You apply to Your Copyright
   and Similar Rights in Your contributions to Adapted Material in
   accordance with the terms and conditions of this Public License.

c. BY-SA Compatible License means a license listed at
   creativecommons.org/compatiblelicenses, approved by Creative
   Commons as essentially the equivalent of this Public License.

d. Copyright and Similar Rights means copyright and/or similar rights
   closely related to copyright including, without limitation,
   performance, broadcast, sound recording, and Sui Generis Database
   Rights, without regard to how the rights are labeled or
   categorized. For purposes of this Public License, the rights
   specified in Section 2(b)(1)-(2) are not Copyright and Similar
   Rights.

e. Effective Technological Measures means those measures that, in the
   absence of proper authority, may not be circumvented under laws
   fulfilling obligations under Article 11 of the WIPO Copyright
   Treaty adopted on December 20, 1996, and/or similar international
   agreements.

f. Exceptions and Limitations means fair use, fair dealing, and/or
   any other exception or limitation to Copyright and Similar Rights
   that applies to Your use of the Licensed Material.

g. License Elements means the license attributes listed in the name
   of a Creative Commons Public License. The License Elements of this
   Public License are Attribution and ShareAlike.

h. Licensed Material means the artistic or literary work, database,
   or other material to which the Licensor applied this Public
   License.

i. Licensed Rights means the rights granted to You subject to the
   terms and conditions of this Public License, which are limited to
   all Copyright and Similar Rights that apply to Your use of the
   Licensed Material and that the Licensor has authority to license.

j. Licensor means the individual(s) or entity(ies) granting rights
   under this Public License.

k. Share means to provide material to the public by any means or
   process that requires permission under the Licensed Rights, such
   as reproduction, public display, public performance, distribution,
   dissemination, communication, or importation, and to make material
   available to the public including in ways that members of the
   public may access the material from a place and at a time
```

(continues on next page)

(continued from previous page)

```
     individually chosen by them.

  l. Sui Generis Database Rights means rights other than copyright
     resulting from Directive 96/9/EC of the European Parliament and of
     the Council of 11 March 1996 on the legal protection of databases,
     as amended and/or succeeded, as well as other essentially
     equivalent rights anywhere in the world.

  m. You means the individual or entity exercising the Licensed Rights
     under this Public License. Your has a corresponding meaning.


Section 2 -- Scope.

  a. License grant.

       1. Subject to the terms and conditions of this Public License,
          the Licensor hereby grants You a worldwide, royalty-free,
          non-sublicensable, non-exclusive, irrevocable license to
          exercise the Licensed Rights in the Licensed Material to:

            a. reproduce and Share the Licensed Material, in whole or
               in part; and

            b. produce, reproduce, and Share Adapted Material.

       2. Exceptions and Limitations. For the avoidance of doubt, where
          Exceptions and Limitations apply to Your use, this Public
          License does not apply, and You do not need to comply with
          its terms and conditions.

       3. Term. The term of this Public License is specified in Section
          6(a).

       4. Media and formats; technical modifications allowed. The
          Licensor authorizes You to exercise the Licensed Rights in
          all media and formats whether now known or hereafter created,
          and to make technical modifications necessary to do so. The
          Licensor waives and/or agrees not to assert any right or
          authority to forbid You from making technical modifications
          necessary to exercise the Licensed Rights, including
          technical modifications necessary to circumvent Effective
          Technological Measures. For purposes of this Public License,
          simply making modifications authorized by this Section 2(a)
          (4) never produces Adapted Material.

       5. Downstream recipients.

            a. Offer from the Licensor -- Licensed Material. Every
               recipient of the Licensed Material automatically
               receives an offer from the Licensor to exercise the
               Licensed Rights under the terms and conditions of this
               Public License.

            b. Additional offer from the Licensor -- Adapted Material.
               Every recipient of Adapted Material from You
               automatically receives an offer from the Licensor to
```

(continues on next page)

(continued from previous page)

```
               exercise the Licensed Rights in the Adapted Material
               under the conditions of the Adapter's License You apply.

            c. No downstream restrictions. You may not offer or impose
               any additional or different terms or conditions on, or
               apply any Effective Technological Measures to, the
               Licensed Material if doing so restricts exercise of the
               Licensed Rights by any recipient of the Licensed
               Material.

      6. No endorsement. Nothing in this Public License constitutes or
         may be construed as permission to assert or imply that You
         are, or that Your use of the Licensed Material is, connected
         with, or sponsored, endorsed, or granted official status by,
         the Licensor or others designated to receive attribution as
         provided in Section 3(a)(1)(A)(i).

  b. Other rights.

      1. Moral rights, such as the right of integrity, are not
         licensed under this Public License, nor are publicity,
         privacy, and/or other similar personality rights; however, to
         the extent possible, the Licensor waives and/or agrees not to
         assert any such rights held by the Licensor to the limited
         extent necessary to allow You to exercise the Licensed
         Rights, but not otherwise.

      2. Patent and trademark rights are not licensed under this
         Public License.

      3. To the extent possible, the Licensor waives any right to
         collect royalties from You for the exercise of the Licensed
         Rights, whether directly or through a collecting society
         under any voluntary or waivable statutory or compulsory
         licensing scheme. In all other cases the Licensor expressly
         reserves any right to collect such royalties.


Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the
following conditions.

  a. Attribution.

      1. If You Share the Licensed Material (including in modified
         form), You must:

            a. retain the following if it is supplied by the Licensor
               with the Licensed Material:

                  i. identification of the creator(s) of the Licensed
                     Material and any others designated to receive
                     attribution, in any reasonable manner requested by
                     the Licensor (including by pseudonym if
                     designated);
```

(continues on next page)

(continued from previous page)

```
                ii. a copyright notice;

               iii. a notice that refers to this Public License;

                iv. a notice that refers to the disclaimer of
                    warranties;

                 v. a URI or hyperlink to the Licensed Material to the
                    extent reasonably practicable;

            b. indicate if You modified the Licensed Material and
               retain an indication of any previous modifications; and

            c. indicate the Licensed Material is licensed under this
               Public License, and include the text of, or the URI or
               hyperlink to, this Public License.

      2. You may satisfy the conditions in Section 3(a)(1) in any
         reasonable manner based on the medium, means, and context in
         which You Share the Licensed Material. For example, it may be
         reasonable to satisfy the conditions by providing a URI or
         hyperlink to a resource that includes the required
         information.

      3. If requested by the Licensor, You must remove any of the
         information required by Section 3(a)(1)(A) to the extent
         reasonably practicable.

  b. ShareAlike.

     In addition to the conditions in Section 3(a), if You Share
     Adapted Material You produce, the following conditions also apply.

      1. The Adapter's License You apply must be a Creative Commons
         license with the same License Elements, this version or
         later, or a BY-SA Compatible License.

      2. You must include the text of, or the URI or hyperlink to, the
         Adapter's License You apply. You may satisfy this condition
         in any reasonable manner based on the medium, means, and
         context in which You Share Adapted Material.

      3. You may not offer or impose any additional or different terms
         or conditions on, or apply any Effective Technological
         Measures to, Adapted Material that restrict exercise of the
         rights granted under the Adapter's License You apply.


Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that
apply to Your use of the Licensed Material:

  a. for the avoidance of doubt, Section 2(a)(1) grants You the right
     to extract, reuse, reproduce, and Share all or a substantial
     portion of the contents of the database;
```

(continues on next page)

(continued from previous page)

```
  b. if You include all or a substantial portion of the database
     contents in a database in which You have Sui Generis Database
     Rights, then the database in which You have Sui Generis Database
     Rights (but not its individual contents) is Adapted Material,

     including for purposes of Section 3(b); and
  c. You must comply with the conditions in Section 3(a) if You Share
     all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not
replace Your obligations under this Public License where the Licensed
Rights include other Copyright and Similar Rights.


Section 5 -- Disclaimer of Warranties and Limitation of Liability.

  a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE
     EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS
     AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF
     ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS,
     IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION,
     WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR
     PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS,
     ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT
     KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT
     ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

  b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE
     TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION,
     NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT,
     INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES,
     COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR
     USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN
     ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR
     DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR
     IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

  c. The disclaimer of warranties and limitation of liability provided
     above shall be interpreted in a manner that, to the extent
     possible, most closely approximates an absolute disclaimer and
     waiver of all liability.


Section 6 -- Term and Termination.

  a. This Public License applies for the term of the Copyright and
     Similar Rights licensed here. However, if You fail to comply with
     this Public License, then Your rights under this Public License
     terminate automatically.

  b. Where Your right to use the Licensed Material has terminated under
     Section 6(a), it reinstates:

       1. automatically as of the date the violation is cured, provided
          it is cured within 30 days of Your discovery of the
          violation; or
```

(continues on next page)

(continued from previous page)

```
       2. upon express reinstatement by the Licensor.

     For the avoidance of doubt, this Section 6(b) does not affect any
     right the Licensor may have to seek remedies for Your violations
     of this Public License.

  c. For the avoidance of doubt, the Licensor may also offer the
     Licensed Material under separate terms or conditions or stop
     distributing the Licensed Material at any time; however, doing so
     will not terminate this Public License.

  d. Sections 1, 5, 6, 7, and 8 survive termination of this Public
     License.


Section 7 -- Other Terms and Conditions.

  a. The Licensor shall not be bound by any additional or different
     terms or conditions communicated by You unless expressly agreed.

  b. Any arrangements, understandings, or agreements regarding the
     Licensed Material not stated herein are separate from and
     independent of the terms and conditions of this Public License.


Section 8 -- Interpretation.

  a. For the avoidance of doubt, this Public License does not, and
     shall not be interpreted to, reduce, limit, restrict, or impose
     conditions on any use of the Licensed Material that could lawfully
     be made without permission under this Public License.

  b. To the extent possible, if any provision of this Public License is
     deemed unenforceable, it shall be automatically reformed to the
     minimum extent necessary to make it enforceable. If the provision
     cannot be reformed, it shall be severed from this Public License
     without affecting the enforceability of the remaining terms and
     conditions.

  c. No term or condition of this Public License will be waived and no
     failure to comply consented to unless expressly agreed to by the
     Licensor.

  d. Nothing in this Public License constitutes or may be interpreted
     as a limitation upon, or waiver of, any privileges and immunities
     that apply to the Licensor or You, including from the legal
     processes of any jurisdiction or authority.


=======================================================================

Creative Commons is not a party to its public
licenses. Notwithstanding, Creative Commons may elect to apply one of
its public licenses to material it publishes and in those instances
will be considered the “Licensor.” The text of the Creative Commons
public licenses is dedicated to the public domain under the CC0 Public
Domain Dedication. Except for the limited purpose of indicating that
```

(continues on next page)

(continued from previous page)

```
material is shared under a Creative Commons public license or as
otherwise permitted by the Creative Commons policies published at
creativecommons.org/policies, Creative Commons does not authorize the
use of the trademark "Creative Commons" or any other trademark or logo
of Creative Commons without its prior written consent including,
without limitation, in connection with any unauthorized modifications
to any of its public licenses or any other arrangements,
understandings, or agreements concerning use of licensed material. For
the avoidance of doubt, this paragraph does not form part of the
public licenses.

Creative Commons may be contacted at creativecommons.org.
```

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

X

Y