Pacemaker Development

Working with the Pacemaker Code Base

Andrew Beekhof

Ken Gaillot

Pacemaker Development
Working with the Pacemaker Code Base
Edition 2

Author	Andrew Beekhof	andrew@beekhof.net
Author	Ken Gaillot	kgaillot@redhat.com

This document has guidelines and tips for developers interested in editing Pacemaker source code and submitting changes for inclusion in the project.

[1] An explanation of CC-BY-SA is available at *https://creativecommons.org/licenses/by-sa/4.0/*

Table of Contents

Frequently Asked Questions

Q: Who is this document intended for?

A: Anyone who wishes to read and/or edit the Pacemaker source code. Casual contributors should feel free to read just this FAQ, and consult other sections as needed.

Q: Where is the source code for Pacemaker?

A: The *source code for Pacemaker*[1] is kept on *GitHub*[2], as are all software projects under the *ClusterLabs*[3] umbrella. Pacemaker uses *Git*[4] for source code management. If you are a Git newbie, the *gittutorial(7) man page*[5] is an excellent starting point. If you're familiar with using Git from the command line, you can create a local copy of the Pacemaker source code with: **`git clone https://github.com/ClusterLabs/pacemaker.git pacemaker`**

Q: What are the different Git branches and repositories used for?

A:

- The *master branch*[6] is the primary branch used for development.
- The *1.1 branch*[7] contains the latest official release, and normally does not receive any changes. During the release cycle, it will contain release candidates for the next official release, and will receive only bug fixes.
- The *1.0 repository*[8] is a frozen snapshot of the 1.0 release series, and is no longer developed.
- Messages will be posted to the *developers@clusterlabs.org*[9] mailing list during the release cycle, with instructions about which branches to use when submitting requests.

Q: How do I build from the source code?

A: See *INSTALL.md*[10] in the main checkout directory.

Q: What coding style should I follow?

A: You'll be mostly fine if you simply follow the example of existing code. When unsure, see the relevant section of this document for language-specific recommendations. Pacemaker has grown and evolved organically over many years, so you will see much code that doesn't conform to the current guidelines. We discourage making changes solely to bring code into conformance, as any change requires developer time for review and opens the possibility of adding bugs. However, new code should follow the guidelines, and it is fine to bring lines of older code into conformance when modifying that code for other reasons.

[1] https://github.com/ClusterLabs/pacemaker
[2] https://github.com/
[3] https://github.com/ClusterLabs
[4] https://git-scm.com/
[5] http://schacon.github.io/git/gittutorial.html
[6] https://github.com/ClusterLabs/pacemaker/tree/master
[7] https://github.com/ClusterLabs/pacemaker/tree/1.1
[8] https://github.com/ClusterLabs/pacemaker-1.0
[9] http://clusterlabs.org/mailman/listinfo/developers
[10] https://github.com/ClusterLabs/pacemaker/blob/master/INSTALL.md

Q: How should I format my Git commit messages?

A: See existing examples in the git log. The first line should look like **`change-type: affected-code: explanation`** where **`change-type`** can be **`Fix`** or **`Bug`** for most bug fixes, **`Feature`** for new features, **`Log`** for changes to log messages or handling, **`Doc`** for changes to documentation or comments, or **`Test`** for changes in CTS and regression tests. You will sometimes see **`Low`**, **`Med`** (or **`Mid`**) and **`High`** used instead for bug fixes, to indicate the severity. The important thing is that only commits with **`Feature`**, **`Fix`**, **`Bug`**, or **`High`** will automatically be included in the change log for the next release. The **`affected-code`** is the name of the component(s) being changed, for example, **`pacemaker-controld`** or **`libcrmcommon`** (it's more free-form, so don't sweat getting it exact). The **`explanation`** briefly describes the change. The git project recommends the entire summary line stay under 50 characters, but more is fine if needed for clarity. Except for the most simple and obvious of changes, the summary should be followed by a blank line and then a longer explanation of *why* the change was made.

Q: How can I test my changes?

A: Most importantly, Pacemaker has regression tests for most major components; these will automatically be run for any pull requests submitted through GitHub. Additionally, Pacemaker's Cluster Test Suite (CTS) can be used to set up a test cluster and run a wide variety of complex tests. This document will have more detail on testing in the future.

Q: What is Pacemaker's license?

A: Except where noted otherwise in the file itself, the source code for all Pacemaker programs is licensed under version 2 or later of the GNU General Public License (*GPLv2+*[11]), its headers and libraries under version 2.1 or later of the less restrictive GNU Lesser General Public License (*LGPLv2.1+*[12]), its documentation under version 4.0 or later of the Creative Commons Attribution-ShareAlike International Public License (*CC-BY-SA*[13]), and its init scripts under the *Revised BSD*[14] license. If you find any deviations from this policy, or wish to inquire about alternate licensing arrangements, please e-mail *andrew@beekhof.net*[15]. Licensing issues are also discussed on the *ClusterLabs wiki*[16].

Q: How can I contribute my changes to the project?

A: Contributions of bug fixes or new features are very much appreciated! Patches can be submitted as *pull requests*[17] via GitHub (the preferred method, due to its excellent *features*[18]), or e-mailed to the *developers@clusterlabs.org*[19] mailing list as an attachment in a format Git can import.

Q: What if I still have questions?

[11] https://www.gnu.org/licenses/gpl-2.0.html
[12] https://www.gnu.org/licenses/lgpl-2.1.html
[13] https://creativecommons.org/licenses/by-sa/4.0/legalcode
[14] https://opensource.org/licenses/BSD-3-Clause
[15] mailto:andrew@beekhof.net
[16] http://clusterlabs.org/wiki/License
[17] https://help.github.com/articles/using-pull-requests/
[18] https://github.com/features/
[19] http://clusterlabs.org/mailman/listinfo/developers

A: Ask on the *developers@clusterlabs.org*[20] mailing list for development-related questions, or on the *users@clusterlabs.org*[21] mailing list for general questions about using Pacemaker. Developers often also hang out on *freenode's*[22] #clusterlabs IRC channel.

[20] http://clusterlabs.org/mailman/listinfo/developers
[21] http://clusterlabs.org/mailman/listinfo/users
[22] http://freenode.net/

C Coding Guidelines

Table of Contents

2.1. C Boilerplate

Every C file should start like this:

```
/*
 * Copyright <YYYY[-YYYY]> Andrew Beekhof <andrew@beekhof.net>
 *
 * This source code is licensed under <LICENSE> WITHOUT ANY WARRANTY.
 */
```

<YYYY> is the year the code was *originally* created. [1] If the code is modified in later years, add **-YYYY** with the most recent year of modification.

<LICENSE> should follow the policy set forth in the ***COPYING***[2] file, generally one of "GNU General Public License version 2 or later (GPLv2+)" or "GNU Lesser General Public License version 2.1 or later (LGPLv2.1+)".

2.2. Formatting

2.2.1. Whitespace

- Indentation must be 4 spaces, no tabs.
- Do not leave trailing whitespace.

[1] See the U.S. Copyright Office's *"Compendium of U.S. Copyright Office Practices"* [https://www.copyright.gov/comp3/], particularly "Chapter 2200: Notice of Copyright", sections 2205.1(A) and 2205.1(F), or *"Updating Copyright Notices"* [https://techwhirl.com/updating-copyright-notices/] for a more readable summary.
[2] https://github.com/ClusterLabs/pacemaker/blob/master/COPYING

2.2.2. Line Length

- Lines should be no longer than 80 characters unless limiting line length significantly impacts readability.

2.2.3. Pointers

- The ***** goes by the variable name, not the type:

```
char *foo;
```

- Use a space before the ***** and after the closing parenthesis in a cast:

```
char *foo = (char *) bar;
```

2.2.4. Functions

- In the function definition, put the return type on its own line, and place the opening brace by itself on a line:

```
static int
foo(void)
{
```

- For functions with enough arguments that they must break to the next line, align arguments with the first argument:

```
static int
function_name(int bar, const char *a, const char *b,
              const char *c, const char *d)
{
```

- If a function name gets really long, start the arguments on their own line with 8 spaces of indentation:

```
static int
really_really_long_function_name_this_is_getting_silly_now(
        int bar, const char *a, const char *b,
        const char *c, const char *d)
{
```

2.2.5. Control Statements (if, else, while, for, switch)

- The keyword is followed by one space, then left parenthesis without space, condition, right parenthesis, space, opening bracket on the same line. **`else`** and **`else if`** are on the same line with the ending brace and opening brace, separated by a space:

```
if (condition1) {
    statement1;
} else if (condition2) {
    statement2;
```

```
} else {
    statement3;
}
```

- In a **`switch`** statement, **`case`** is indented one level, and the body of each **`case`** is indented by another level. The opening brace is on the same line as **`switch`**.

```
switch (expression) {
    case 0:
        command1;
        break;
    case 1:
        command2;
        break;
    default:
        command3;
}
```

2.2.6. Operators

- Operators have spaces from both sides. Do not rely on operator precedence; use parentheses when mixing operators with different priority.
- No space is used after opening parenthesis and before closing parenthesis.

```
x = a + b - (c * d);
```

2.3. Naming Conventions

- Any exposed symbols in libraries (non-**`static`** function names, type names, etc.) must begin with a prefix appropriate to the library, for example, **`crm_`**, **`pe_`**, **`st_`**, **`lrm_`**.

2.4. vim Settings

Developers who use **`vim`** to edit source code can add the following settings to their **`~/.vimrc`** file to follow Pacemaker C coding guidelines:

```
" follow Pacemaker coding guidelines when editing C source code files
filetype plugin indent on
au FileType c   setlocal expandtab tabstop=4 softtabstop=4 shiftwidth=4 textwidth=80
autocmd BufNewFile,BufRead *.h set filetype=c
let c_space_errors = 1
```

Python Coding Guidelines

Table of Contents

3.1. Python Boilerplate

If a Python file is meant to be executed (as opposed to imported), it should have a **`.in`** extension, and its first line should be:

```
#!@PYTHON@
```

which will be replaced with the appropriate python executable when Pacemaker is built. To make that happen, add an AC_CONFIG_FILES() line to configure.ac, and add the file name without .in to .gitignore (see existing examples).

After the above line if any, every Python file should start like this:

```
""" <BRIEF-DESCRIPTION>
"""

# Pacemaker targets compatibility with Python 2.7 and 3.2+
from __future__ import print_function, unicode_literals, absolute_import, division

__copyright__ = "Copyright <YYYY[-YYYY]> Andrew Beekhof <andrew@beekhof.net>"
__license__ = "<LICENSE> WITHOUT ANY WARRANTY"
```

If the file is meant to be directly executed, the first line (**<SHEBANG>**) should be **`#!/usr/bin/python`**. If it is meant to be imported, omit this line.

<BRIEF-DESCRIPTION> is obviously a brief description of the file's purpose. The string may contain any other information typically used in a Python file *docstring*[1].

The **`import`** statement is discussed further in *Section 3.2.1, "Python Future Imports"*.

<YYYY> is the year the code was *originally* created. [2] If the code is modified in later years, add **`-YYYY`** with the most recent year of modification.

[1] https://www.python.org/dev/peps/pep-0257/
[2] See the U.S. Copyright Office's *"Compendium of U.S. Copyright Office Practices"* [https://www.copyright.gov/comp3/], particularly "Chapter 2200: Notice of Copyright", sections 2205.1(A) and 2205.1(F), or *"Updating Copyright Notices"* [https://techwhirl.com/updating-copyright-notices/] for a more readable summary.

<LICENSE> should follow the policy set forth in the ***COPYING***[3] file, generally one of "GNU General Public License version 2 or later (GPLv2+)" or "GNU Lesser General Public License version 2.1 or later (LGPLv2.1+)".

3.2. Python Compatibility

Pacemaker targets compatibility with Python 2.7, and Python 3.2 and later. These versions have added features to be more compatible with each other, allowing us to support both the 2 and 3 series with the same code. It is a good idea to test any changes with both Python 2 and 3.

3.2.1. Python Future Imports

The future imports used in *Section 3.1, "Python Boilerplate"* mean:

- All print statements must use parentheses, and printing without a newline is accomplished with the **`end=' '`** parameter rather than a trailing comma.
- All string literals will be treated as Unicode (the **`u`** prefix is unnecessary, and must not be used, because it is not available in Python 3.2).
- Local modules must be imported using **`from . import`** (rather than just **`import`**). To import one item from a local module, use **`from .modulename import`** (rather than **`from modulename import`**).
- Division using **`/`** will always return a floating-point result (use **`//`** if you want the integer floor instead).

3.2.2. Other Python Compatibility Requirements

- When specifying an exception variable, always use **`as`** instead of a comma (e.g. **`except Exception as e`** or **`except (TypeError, IOError) as e`**). Use **`e.args`** to access the error arguments (instead of iterating over or subscripting **`e`**).
- Use **`in`** (not **`has_key()`**) to determine if a dictionary has a particular key.
- Always use the I/O functions from the **`io`** module rather than the native I/O functions (e.g. **`io.open()`** rather than **`open()`**).
- When opening a file, always use the **`t`** (text) or **`b`** (binary) mode flag.
- When creating classes, always specify a parent class to ensure that it is a "new-style" class (e.g. **`class Foo(object):`** rather than **`class Foo:`**)
- Be aware of the bytes type added in Python 3. Many places where strings are used in Python 2 use bytes or bytearrays in Python 3 (for example, the pipes used with **`subprocess.Popen()`**). Code should handle both possibilities.
- Be aware that the **`items()`**, **`keys()`**, and **`values()`** methods of dictionaries return lists in Python 2 and views in Python 3. In many case, no special handling is required, but if the code needs to use list methods on the result, cast the result to list first.
- Do not raise or catch strings as exceptions (e.g. **`raise "Bad thing"`**).

[3] https://github.com/ClusterLabs/pacemaker/blob/master/COPYING

- Do not use the **`cmp`** parameter of sorting functions (use **`key`** instead, if needed) or the **`__cmp__()`** method of classes (implement rich comparison methods such as **`__lt__()`** instead, if needed).
- Do not use the **`buffer`** type.
- Do not use features not available in all targeted Python versions. Common examples include:
  - The **`html`**, **`ipaddress`**, and **`UserDict`** modules
  - The **`subprocess.run()`** function
  - The **`subprocess.DEVNULL`** constant
  - **`subprocess`** module-specific exceptions

3.2.3. Python Usages to Avoid

Avoid the following if possible, otherwise research the compatibility issues involved (hacky workarounds are often available):

- long integers
- octal integer literals
- mixed binary and string data in one data file or variable
- metaclasses
- **`locale.strcoll`** and **`locale.strxfrm`**
- the **`configparser`** and **`ConfigParser`** modules
- importing compatibility modules such as **`six`** (so we don't have to add them to Pacemaker's dependencies)

3.3. Formatting Python Code

- Indentation must be 4 spaces, no tabs.
- Do not leave trailing whitespace.
- Lines should be no longer than 80 characters unless limiting line length significantly impacts readability. For Python, this limitation is flexible since breaking a line often impacts readability, but definitely keep it under 120 characters.
- Where not conflicting with this style guide, it is recommended (but not required) to follow *PEP 8*[4].
- It is recommended (but not required) to format Python code such that **`pylint --disable=line-too-long,too-many-lines,too-many-instance-attributes,too-many-arguments,too-many-statements`** produces minimal complaints (even better if you don't need to disable all those checks).

[4] https://www.python.org/dev/peps/pep-0008/

Appendix A. Revision History

Revision 1-0 **Tue Jul 26 2016** **Ken Gaillot** *kgaillot@redhat.com*

Convert coding guidelines and developer FAQ to Publican document

Revision 1-1 **Mon Aug 29 2016** **Ken Gaillot** *kgaillot@redhat.com*

Add Python coding guidelines, and more about licensing

Revision 2-0 **Fri Jan 12 2018** **Ken Gaillot** *kgaillot@redhat.com*

Drop support for Python 2.6

Index

Symbols

B

C

D

F

G

L

M

N

O

P

S

V

W