Software Development on Linux Systems

4002-XXX-XX

By

Cody Van De Mark

Today

- Documentation Overview
- Code Documentation
- Release Documentation

Documentation Overview

- Documentation in open source is tremendously important as people will not use your software if they do not have instructions
- Documentation is made for you, other developers and end user
- Open source projects are expected to have documents for release information, end-users, developers and licensing

Documentation Overview

- Code documentation includes:
  - Code commenting
  - Man pages
  - Architecture/design information
  - API information

- Release documentation
  - Change logs
  - Licensing
  - README file

Code Comments

- You are already aware that code comments are important, but in open source they are even more important
- Open source code should be well commented and kept clean as many other developers will be picking it up with little support otherwise
- You may have thousands of people looking at and modifying your code regularly

Code Comments

- Code comments should not be at every line explaining everything;

  They should be added for code that is not immediately obvious to any programmer

- Your comments should assume the programmer has a base under of the language itself

  IE: Do not explain how Java itself works to them unless you are using an obscure library that modifies the language behavior itself

Code Comments

- Code comments should be at the beginning of methods explaining what the method does and possibly where the parameters are coming from

- Code comments should be put in ahead of algorithms explaining the algorithm and how to modify it

- You may want to explain calls to other methods/classes if it is not obvious why you needed to call that that function

- Takeaway: Comment code that is not obvious to a new developer

Man Pages

- Man pages are very important to the open source community as they provide a standardized reference manual for any application
- Man pages list usage and flag information for end users
- Man pages are intended for Unix users to quickly look up usage information they may have forgotten
- Unix users, system administrators and developers have specific expectations for a man page

Man Pages

- Man pages are expected to look roughly the same as it is a standardized manual format

- Man pages include usage information, arguments (flags), bug information, further information links, etc

- Man pages are very concise, clear references to an application that users can quickly look up when they forget an argument format or are looking for a specific flag, such as “format results as JSON”

Man Pages

- Man pages are broken into several categories, though you will probably only use one or two of them

Category	Description
1	General Commands
2	System Calls
3	Library Functions and subroutines
4	Devices and drivers
5	File formats and convention
6	Games
7	Miscellaneous
8	System Administration and daemons

Man Pages

- Each man page has specific information sections

  Typical Sections:

Category	Description
Name	This is the name of the program and a one list description of it
Synopsis	This is the base syntax on how to invoke the program from the command line
Description	This is a clear thought out description of the program, its functionality and purpose
Options	This is a list of parameters and flags that the program can take along with what they do
Bugs	This is a URL and/or email that the user can reference to report bugs and find bugs. This may also be a list of known bugs.
See Also	This is a list of other documentation (such as html manuals/tutorials) and related programs the user may be interested in.

Man Pages

- You may also add sections to the man pages for other necessary info
- Other sections you may find are:
    - Author
    - Version
    - Known bugs
    - History
    - Examples
    - Files
    - Exit status
    - Environment

Man Pages

Syntax:

.TH [program name] [section number] [center footer] [left footer] [center header]

.SH [section name]
Section text

.SS [subsection name]
Subsection text

.IP [item name]
itemized text

.B [text to be in bold on this line]

.I [text to be in italic on this line]

Man Pages

- Example Code:

```
.TH hellotest 1 "January 2011" "1.1 Linux Version" "User Manuals"
.SH NAME
hellotest \- Application that prints "hello world"

.SH SYNOPSIS
hellotest [--debug][-n #]

.SH DESCRIPTION
.B hellotest 1.1
hellotest allows you to print “Hello World” from the command. This is done as practice in many
languages as the first application written.

.SH OPTIONS
.IP --debug
Print verbose runtime information to the terminal that can be used for debugging purposes.
```

Man Pages

- Output of Example:

```
hellotest(1)                    User manuals                    hellotest(1)

NAME
hellotest - Application that prints “hello world ”

SYNOPSIS
hellotest [--debug] [-n #]

DESCRIPTION
hellotest 1.1
hellotest allows you to print “Hello World” from the command. This is done as practice in many
languages as the first application written.

OPTIONS
    --debug
        Print verbose runtime information to the terminal that can be used for debugging
            purposes.
```

Architecture Information

- Architecture/design information is another important aspect of documentation in open source

- Developers are picking up your code to improve on it;

  You want to help them understand your design and the architecture

- Architecture information provides them with an understanding of how the components work together with each other, other software and the system

Architecture Information

- As open source projects can be quite complex

  IE: Eclipse has over 20 million lines of code

- Developers have little idea where to start in any complex project;

  You wouldn't want to start digging through 20 million lines of code in thousands of files trying to find XML parsing code

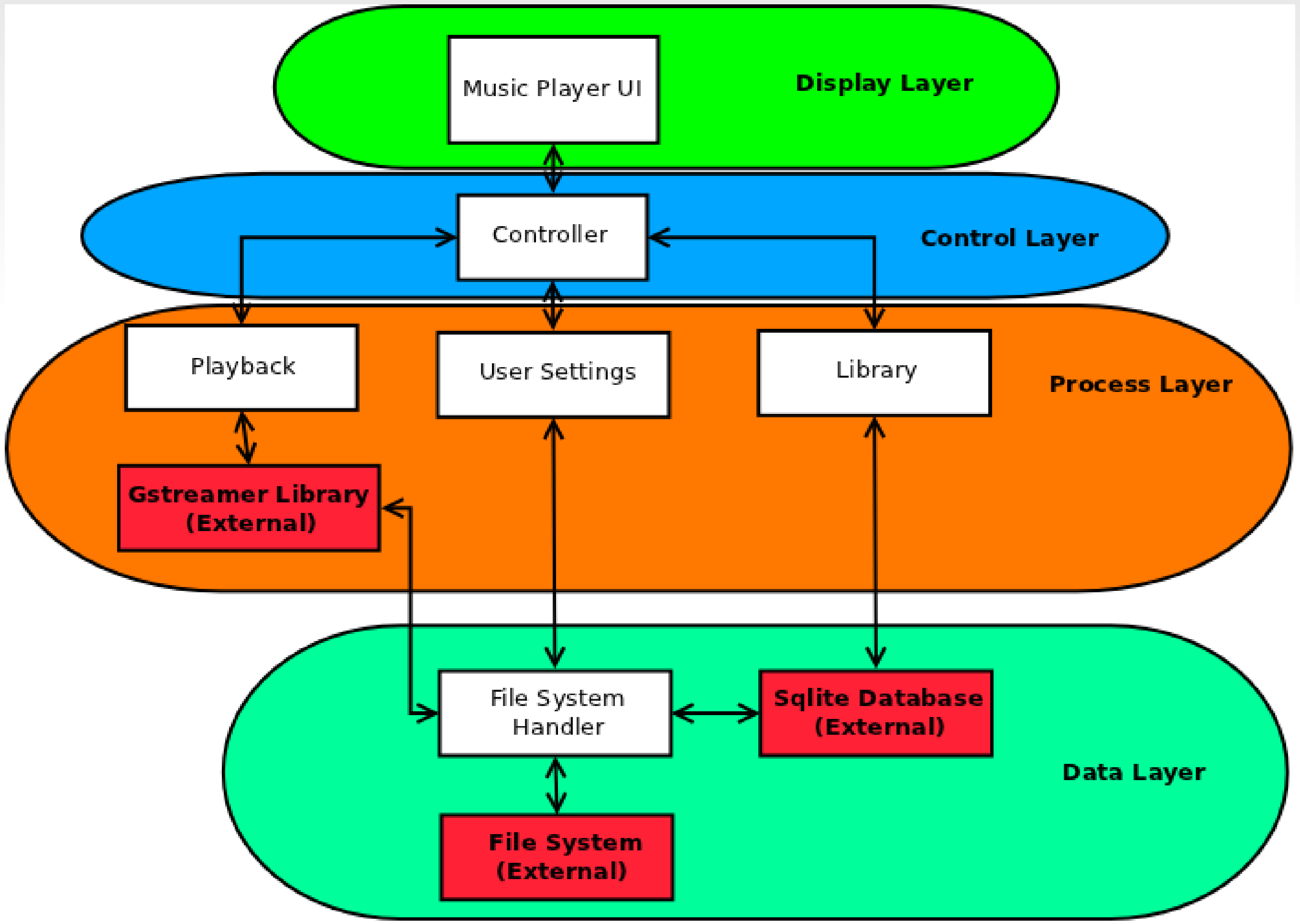
- Architecture information is necessary to allow developers to understand where to find the code they are looking for and how to interact with other components

Architecture Information

- Though architecture information should be supplied in several forms, an architecture diagram is one of the most helpful forms
- Architecture diagrams provide a graphic high level representation of how code interacts between various layers of the project
- Architecture diagrams are fairly high-level, but should also be provided in lower level, more detailed forms

Architecture Information

- Architecture Diagram Example:



Architecture Information

- The architecture diagram example showed how each layer communicated through each component, and which external components were used

- You should also provide similar architecture diagrams of each layer itself in more detail to show how the components and classes themselves communicate

- The more detail you can **coherently** and **clearly** provide a developer, the easier it will be for them to work on your project

Application Programming Interface

- Your project should have an API that can be used for integration;

  IE: GStreamer media libraries have an API that allows mp3 decoding and playback from any application that calls the API

- API information should be provided for developers who want to integrate and use your code, but not necessarily modify it

- This is very important for developers who are looking for leverage your project, especially if they want to embed your project

Application Programming Interface

- Your API should be well documented outside of the code

  IE: Text documents, html pages, etc

- These documents should tell what the functions are publicly available and what arguments they take

- They should also provide an explanation of each function, what it will return (if anything) and the format of what will be returned

- These documents should also give any information developers should known when trying to call your methods

Change Logs

- Change logs are a list of recent changes to your software by release version
- Change logs grow on previous change logs, almost always documenting back to the original release
- Change logs are intended to provide end-users and developers a list of changes, such as new features, removed features, fixed bugs, etc
- Change logs are formatted different between Debian systems and RedHat systems

Change Logs

- Ubuntu and other Debian systems follow the Debian format and are extremely strict

Tag	Purpose	Options
Package	Software name	Name
version	Version number	Number(s)
distribution	Package type (determined by team)	Experimental, stable, stable-security, testing-proposed-updates, testing-security, unreleased, unstable, ***name*** ****name*** may be a release name the team have chosen, such as penguin, coyote, fossil, etc
urgency	Severity type (determined by team)	Low, medium, high, emergency, critical *emergency and critical are treated the same
maintainer-name	Real name or pseudonym	Real name or pseudonym
date	Date in RFC 2822 and RFC 5322 format	Day-of-week, dd month yyyy hh:mm:ss +**zzzz** ***zzzz** is the time zone in UTC time which is HHMM. In New York this is **-0500**

Change Logs

- Debian Format Example:

```
hellotest (1.1) unstable; urgency=low

  *Two spaces then an astrix
    four space then more information
  *Added new music
    New songs are owned by the team and free to use
    Playback can be done in mono or stereo

 -- MaintainerGuy <maintainer@mGuy.it.com.edu>  Fri, 12 Jan 2012 14:23:20  -0500

hellotest (1.0) stable; urgency=medium

  *Modified code to prevent sporadic crashes
    Requires gcc (<= 4.5.2)
    Requires lib-sdl (<= 1.2.14)

 -- MaintainerGuy <maintainer@mGuy.it.com.edu>  Mon, 20 Dec 2011 15:50:43  -0500
```

Change Logs

- Debian Format Example with Spacing:

```
hellotest (1.1) unstable; urgency=low

■■*Two spaces then an astrix
■■■■four space then more information
■■*Added new music
■■■■New songs are owned by the team and free to use
■■Playback can be done in mono or stereo

■-- MaintainerGuy <maintainer@mGuy.it.com.edu>  Fri, 12 Jan 2012 14:23:20  -0500

hellotest (1.0) stable; urgency=medium

■■*Modified code to prevent sporadic crashes
■■■■Requires gcc (<= 4.5.2)
■■■■Requires lib-sdl (<= 1.2.14)

■-- MaintainerGuy <maintainer@mGuy.it.com.edu>  Mon, 20 Dec 2011 15:50:43  -0500
```

Change Logs

- Fedora follows the Fedora format and is less strict than Debian

- Format

```
* day-of-week mmmm dd yyyy Name <email> - [version]
- comment
- comment
```

- Fedora Format with Spacing

```
*▮day-of-week mmmm dd yyyy Name <email> - [version]
-▮comment
-▮comment
```

Change Logs

- Fedora Example:

```
* Thu Jan 19 2012 MaintainerGuy <maintainer@mGuy.it.com.edu> 1.1
- New version released
- Fixed compile errors on Linux Kernel 2.8

* Wed Jan 18 2012 MaintainerGuy <maintainer@mGuy.it.com.edu> 1.0
- Modified code to prevent sporadic crashes
- Removed midi sound component
```

- Fedora Example with Spacing:

```
*▮Thu Jan 19 2012 MaintainerGuy <maintainer@mGuy.it.com.edu> 1.1
-▮New version released
-▮Fixed compile errors on Linux Kernel 2.8

*▮Wed Jan 18 2012 MaintainerGuy <maintainer@mGuy.it.com.edu> 1.0
-▮Modified code to prevent sporadic crashes
-▮Removed midi sound component
```

Licensing

- As we have talked about, you must attach a license with your code, especially if you are not the owner or the code is a licensed project
- Different distributions have different formats for licensing
- Ubuntu and other Debian distributions follow the Debian format
- Fedora and other RedHat Distributions follow the Fedora format
- The Debian format is very strict, while the Fedora format is much less so

Licensing

- The Debian format

```
Format: http://dep.debian.net/deps/dep5
Upstream-Name: hello
Source: http://example.com

Files: *
Copyright:  2012  programmerGuy <programmerGuy@guy.net.edu>
            2012 programmerFriend  <programmerFriend@guy.net.edu>

License: GPL-2+
 License text indented 1 space on each line.
 More license text indented 1 space on each line

Files: debian/*
Copyright:  2012  programmerGuy  <programmerGuy@guy.net.edu>

License: GPL-2+
 License text indented 1 space on each line
 More license text indented 1 space on each line
```

Licensing

- For the Fedora format, take a copy of the actual software license from a website or your system and rename it to COPYING

- The Fedora format is specified within the RPM package SPEC file on the %doc line

  Example:

  ```
  License: GPL
  %doc COPYING
  ```

Licensing

- In addition to the license itself, your code needs to have the license header in each file (.c, .cpp, .java, .py, etc)
- The license header includes the name of the file, what it is, the copyright, the author and the license heading
- This quickly references the license of that file in projects that may be leverage existing code/libraries from many different licenses

Licensing

- This is extremely important!
- If you download a proprietary library, that happens to have GPL code in it and you modify the GPL files, you must legally redistribute the GPL files as GPL
- The license header in each file allows you to track which ones may be licensed to different licenses and what your rights are
- Many open source projects contain files under many licenses, and your changes to those files must follow the license it is under

Licensing

- Example License Header: * From Secret Maryo Chronicles

```
/***********************************************************************************
* player.cpp  -  Level player class
*
* Copyright (C) 2003 - 2009 Florian Richter
***********************************************************************************/
/*
        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 the Free Software Foundation; either version 3
        of the License, or (at your option) any later version.

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

#include "../core/globals.h"
#include "../player/player.h"

namespace SMC
{
```

README

- README files contain
  - End user instructions
  - Website URLs
  - Help URLS
  - Configuration information
  - Setup information
  - General information

README

- README files are intended to provide users with a starting point when they first download your software
- You want to include information end users need, such as
  - Descriptions
  - Controls
  - Contact information
  - Customization options
  - Configuration file information
  - Basic setup information
  - Etc

README

- Example of Simplified README file:

```
Onboard 0.94.0
------------
Description:
Onboard is an onscreen keyboard useful for everybody that cannot use a hardware keboard; for
example TabletPC users, mobility impaired users,...It has been designed with simplicity in mind and
can be used right away without the need of any configuration, as it can read the keyboard layout
from the X server. Users can nevertheless define custom layouts.

Among its features are:
- Support of custom layouts through the use of xml and svg files.
- Support for <modifier>+<mouseclick> combination.
- Toggling mouse buttons to perform right clicks with the left mouse button.
- Minimizing the keyboard to the panel, a trayicon, or a floating icon.
- Rudimentary support for scanning.

Homepage:                                Reporting Bugs:
https://launchpad.net/onboard            https://bugs.launchpad.net/onboard

License:
This program is released under the terms of the GNU General
Public License. Please see the file COPYING for details.
```