

Open Source Code Guide

Writing code from scratch isn't always practical, especially when time isn't on your side. So before you start working on your next project, take a look at Elecia's approach to code collection. Working with free code can cut in half the time you'd otherwise spend on the software end of a project.

You're a developer, not an evangelist. Can you use free software? If so, what can you use? Is it any good? Will it save time?

The bottom line is all you really care about, so let's start there. If you're developing commercial, proprietary code, don't use source with a GNU general public license (GPL): it will taint your code, making it GPL as well. Other options are available. The Mozilla public license (MPL) allows you to use whole source files and link them in. If you can find it, you can use and abuse public domain code to your heart's content. Usable code is out there, and it's worth taking a look before rewriting the wheel.

STOP REPEATING

Have you ever written code twice? There has been a fair amount of reiteration in my life as a firmware engineer: PID controllers, various serial

drivers, filters, and tools for image and audio processing. It would be nice to skip the repetition and get on with making gizmos.

This isn't legal advice. You should see a lawyer for that, particularly if your multimillion-dollar company wants to use code without paying for it. In this article I'll describe what I found. Refer to the references at the end of this article if you want to investigate further.

I have never paid much attention to free software or open source. I like Linux, but I don't maintain a system because so many of the embedded compilers are made for Windows. I'm not enough of a fanatic to port things or even buck the trend at a company.

A war's been raging. Don't confuse free software and open source. They aren't the same. They refer to the same thing, but guerilla warriors use the former and the latter is for is the

regular militia. More seriously, the definitions are fluid; they change with the context. For this article, free software means GPL. Open source means anything less restricted.

There are a number of public licenses, and wading through the alphabet soup is a daunting task. Table 1 is an overview of the major licenses and their properties. This article is focused on source code, not the resulting program. Programs developed under public license can be used to make a proprietary product. Their free executables (and their free source) can be distributed along with your proprietary ones. However, reading their source code is like reading a novel: only a line or two can be quoted without infringing on the copyright. There are many more open/free licenses out there. Refer to www.opensource.org/licenses for a comprehensive list.

COPYLEFT

The GPL gives you three rights: the right to copy the software and give it away; the right to change the software; and the right to access the source code.^[1] You must pass on these rights, unimpaired, to other users. Modifications must be distributed in the form of source code as well.

The idea that GPL programs have freely available source code is a big lure. There's more to it than that. The source can be modified and the result used (not distributed!) without you

License	Can be mixed with software that isn't free	Modifications can be taken privately	License can be changed	Requires credit in documentation
Free software: GPL	No	No	No	No
Free software library: LGPL	Yes	No	No	Yes
Open source: MPL	Yes	Yes	No	No
Free BSD	Yes	Yes	No	Yes
Public domain	Yes	Yes	Yes	No

Table 1—These are the most popular licenses. Each license has its own set of requirements. Finding source code with a usable license can be difficult, but it isn't impossible. (Source: D. Ravicher, "Facilitating Collaborative Software Development: The Enforceability of Mass-Market Public Software Licenses," Virginia Journal of Law and Technology, Fall 2000.)

Day	1	2	3	4	5	6	7	8	9	10
Normal development	Plan		Design		Code				Debug	
Searching with no results	Plan	Look	Plan	Design	Code				Debug	
Searching with some results	Plan	Look	Evaluate	Design	Code				Debug	
Searching with good results	Plan	Look	Evaluate	Adapt and debug						

Table 2—Elicia, please write a 2- or 3-sentence caption.

making the modification public. For example, if I were to modify a Linux kernel to make a robot (yes, that would be silly) that simply serves me French press coffee at precisely 6:15 a.m., no one would need to know. However, if I were to sell the robot, the source would have to be available to the public. On the other hand, if I were to leave Linux as it is and use the GNU compiler tools to write a program, I would only have to redistribute any kernel level drivers I made or modified. My application code is my own.

The license says that code is free—free as in freedom and liberty, not free as in beer and pizza. Free software is a concept and a community (think “free society”). Members have access to ample well-written, well-documented code (as well as plenty of dross). However, you must join. Your code also must be free under their definition. Anyone who releases the program (modified or unmodified) must satisfy the restrictions in the GPL that prevent the program and its derivatives from becoming proprietary.

Some people call this the GPL virus or taint. If you use a piece of GPLed code and then compile and distribute the resulting program, all the source code linked together to make the program becomes legally GPL.^[2]

Copyleft is what GPL users call their licensing. It isn’t all that different from copyright. The people who wrote the code own it. If you tinker, you have to follow the rules, and the rules say to pass it on. If you fail to follow the rules, the code reverts to being copyrighted and you still can’t follow through on your nefarious plan.

In exchange for their work, the developers want their principles main-

tained. If you’re willing to trade your intellectual property for theirs, then you’ve joined and everyone gets to play on a level playing field. However, if you don’t want to sign up for this bargain, you’re welcome to develop your own code. Either way, they’ve nicely agreed to let the public use their executable program.

There is another aspect to GPL: the GNU Library, or the “Lesser” general public license (LGPL). The license is intended to be less restrictive to encourage the use of certain libraries (i.e., the GNU C library). Although modifications to the library source code must be made available in source form, you only need to provide object code. You must be able to recreate the executable. You must give “prominent notice” when the LGPL library is used.^[3] Few companies unwilling to free their source code will be willing to adhere to these strict guidelines instead.

MOZILLA PUBLIC LICENSE

Netscape set its code free in 1998. The result was Mozilla. Netscape’s flagship product is a web browser called Firefox. After a few months of use, I’ve found it to be smaller, faster, more secure, and more customizable than Internet Explorer. It has played nicely with almost all of the web sites I’ve used. Firefox is pretty nifty for something I didn’t pay for and could scrutinize for security holes if I were so inclined.

The code developed at Mozilla falls under the MPL. The definition of free is different from the GPL definition. MPL code is available for use and integration, but all modifications must be made public. A modification is defined as any change you make to a

file: an addition, a deletion, and cutting and pasting. If you can use an entire file without modification, you can use this code with impunity. However, if you copy the code into yours, or if you modify the file, it must be made public with MPL licensing.

Refer to the Mozilla Organization’s web site for more information. As you’ll see, there are many uses for MPL code:

The goal of the MPL is to encourage as much innovation as possible. We anticipate that people will take the code licensed under the MPL and combine it with code developed or licensed under other terms. We also anticipate that the combined code will be licensed under a variety of terms, including different payment terms, support terms and use restrictions.^[4]

When Netscape opted to avoid the GPL route, they noted that commercial developers would need to look closely at the legal ramifications of free source code. They believed that there was a barrier that kept companies from working within GPL’s free society. They wanted to remove the barrier with a less restrictive license. Thus, they drafted MPL with the help of a team of lawyers.

Executables made from partially or entirely MPL source code can be distributed with the license of your choosing, even a completely proprietary, closed copyright. Thus, MPL is significantly less viral than GPL, but it isn’t entirely untainted. The code must be kept separate. There are systems where this would be fine and others where this would be a deal

breaker.

Note that the Netscape public license (NPL) is another license that's generally used in the same sentence as MPL. NPL is MPL with some extras for Netscape. If entire files are used, the difference doesn't matter. If not, well, find a lawyer.

FreeBSD

One repository of almost public domain code is under FreeBSD licensing, which is extremely nonrestrictive. It allows you to do almost anything to the code covered in the license, but it requires reference to the copyright holder in the documentation accompanying the software (or product). Older versions of BSD licensing require a mention of the University of California, Berkeley.^[5]

Overall, this isn't a bad deal, but most companies will be squeamish about putting a message in the user manual. What would a large consumer electronics company make of that suggestion by a frontline developer?

PUBLIC DOMAIN

There's a pop psychology question about choosing a superpower: flying or invisibility? Public domain code is a lot like being invisible: you can take what you need, do what you want, and no one's the wiser for it.

Public domain code, like public domain songs, doesn't have a copyright (or patent). It either has run out or the copyright holder has voluntarily relinquished interest. Because copyrights last longer than software, the latter is generally the case. Anyone can use and build on this source code with out restriction. There are sources for public domain code, but they are more difficult to find than the relatively large GPL and MPL repositories.

By design, GPL is incompatible with all licenses that aren't GPL. When I'm trying to find code that can be used as a library for companies not ready to take the free software plunge, the goal is public domain code or an entire MPL file.

EXPLICIT COPYRIGHT

Just because something is on the 'Net without attribution, you don't have the right to use and distribute it. Frequently, the files have no header and the license is in some readme file that they've copied from another project. Because files get separated, you should put copyright notices and terms of use in each and every file. Implicit copyright is dangerous for all involved. To use a file, look for explicit permission to use and distribute the code. I like to know an author's identity so I can offer to buy that person a beer. When I can't find the author and the license isn't explicit, I don't use it.

This is a pretty conservative view of the reusability of the code. I don't want to lose my job and business because of a shortcut.

FINDING CODE

Unless your application is built around code found ahead of time, it will take you some time to find the general desired functionality with the desired license. There is a good chance that the code from the 'Net won't do exactly what's needed. Even if it does, it probably won't drop into your application (different compiler, different chip, etc). Even code that doesn't cost anything isn't free to use. It takes time, which is exactly what you're trying to save.

When you're starting project, how long should you spend looking for code? I'm in favor of spending 5% of my time searching, 10% evaluating, and 20 to 35% adapting and debugging. For example, if I estimate a piece of code will take 10 days to implement, I would spend half a day searching for existing implementations (see Table 2). Hopefully, I'll find two or three. If not, then half of a day will be wasted and I'll have to get to work. If I'm lucky, I'll spend a full day figuring out which (if any) option is good enough to use. Finally, it will take two or three days to put the code in and make it work. This is like any estimate heuristic: it strongly depends on the original estimate and the specific application. Some code will be easy to find and easy to integrate.

Some will look good until the end and then it won't work out.

Using free software is like any programming skill. Practice makes it faster and better. It's important to remember that you are saving some time, not all of it.

CODE REPOSITORIES

SourceForge is the largest repository for free software and open source code. It involves more than 87,000 projects, some of which are active, useful, and good. The licenses here are all over the place: GPL, MPL, BSD, public domain, and a whole bunch of similar ones.

The extended search (available with \$39-per-year subscription) allows you to search by license. However, the packages have multiple boxes checked. Even after filtering out everything but the public domain, much of the code I found was still GPL. However, if GPL code (or any old license) will do, SourceForge is the place to go. It has a rating system, bug tracking, and lots of individuals doing their own thing and occasionally asking for help.

Once of the best embedded systems on SourceForge is the Autopilot project (www.rotomotion.com). It has excellent documentation with a great treatment of how the Kalman filter algorithm works and good code to go along with it. So long as it's under GPL, I'll continue to come back to this one and look at what it does for reference and for my own projects.

Enough about the wonders of SourceForge. My focus isn't only obtaining the code I can use, but also getting code to share with employers who may be hesitant to free their code.

mozilla.org is the obvious place to get MPL code. There aren't any motor controllers, sensor array filters, and device drivers. There are all the bits and pieces for making user applications including a browser, editor, chat program, and mail reader. None of these may seem directly applicable to the embedded world, but they have nice sections—security things such as parsing DSA (digital signature authority), secure hashing, and compression

algorithms (a nice JPEG library).

The code is extremely nice and the code review and bug reporting processes are good. If you need source code to a higher-level application and the MPL restrictions work, Mozilla is a great place to look.

Your chip vendor is the first place to look for public domain code. Many vendors provide code and code generation tools so you'll use their chips. This symbiosis works well for both parties if you take the time to learn what's out there. However, after you've exhausted the obvious choice, there are a number of public domain source code repositories to consider (see Table 3).

Since there isn't any money in public domain code, the repositories are generally outdated. What's available is generally a contribution from an altruist or a student doing homework, or it's from a time when people didn't think software was valuable intellectual property (and before licenses where relatively standardized).

EVALUATING CODE

Getting code that says it does what's needed doesn't mean the code actually does what you need. Generally, the author has worked out a solution to a similar problem. It's a

good first step, but the source still needs to be evaluated, particularly if there are two or three options from which to choose.

Consider the following factors for evaluation. Is it in a language you can use? Can you read the code? (If not, you can't debug it and it isn't a contender.) Does it have adequate comments? (If not, can you understand it and comment it yourself?) Is the algorithm one you want?

The aforementioned questions boil down to one big one: Will it be easier to adapt this code or just write it from scratch? Sometimes it's easier to rewrite bad code than it is to read it. This step is sometimes hard to explain to managers. It's akin to sticking your toe in the water when they want to see splashing. However, if none of the options turn out to be useful, algorithms have been reviewed, examples provided, and niceties remembered. (Oh, yeah, an "emergency stop all" command would be nice.) It will make the design of your own code go faster.

ADAPTING CODE

The next step involves adapting the code. It may be as simple as updating the code to meet style guidelines or as difficult as translating it to a different

language.

You need to consider a few things. Is the documentation up to your standards? Does it meet your style guidelines? Does the code perform proper error checking? Are there RAM considerations? (It may not have been developed for an embedded system.) Are there usable debugging features? If you can only debug with print statements, you may need to add some.

Programmers who use code without reading and understanding it thoroughly get what they deserve when it comes to debugging. And as for debugging, why should the existing code work better the first time than newly written code? The good news is that it probably will. Someone has already used the code. The bad news is that they did something slightly different with it and they might not have been as technically brilliant.

CODE COLLECTION

Using free code takes between 35% and 50% of the time you might otherwise spend developing a piece of code from scratch. It isn't free.

Your chances for finding something will improve if you know what you are looking for and if your project involves heavy-duty science, security, and compression. It's also helpful if

URL	Type	License?	Comments
www.thefreecountry.com	The Free Country is an extensive and up-to-date repository of a variety of code.	Mixed: a fair amount of public domain, the repository comments generally give you a hint.	This site lead to some good places: <ul style="list-style-type: none">• zlib compression/decompression library: www.gzip.org/zlib/• a lovely crypto suite with mostly free code: www.eskimo.com/~weidai/cryptlib.html• standard template library for C++: www.stlport.org/
www.netlib.org	Repository for numerical and scientific packages (matrix handling, sorting, etc.).	Goal is freely available but some is for academic use only.	Code is generally old with lots of Fortran, unsupported with email address that go nowhere. One brilliant exception is CEPHES (http://netlib.org/cephes/), an excellent math package available in C and Fortran.
www.nr.com/public-domain.html	Numerical recipes: Vector matrix allocation (not vector math) and complex number handling in C, C++, and Fortran.	Public domain	Most of the numerical recipes code is proprietary and may not be used without licensing. A small amount has been released to the public domain. Matrix header has a lot of global variables that probably aren't suited for a low RAM embedded environment.
www.gams.nist.gov	Numerical and computational software.	Many public domain, some licensed and some propriety, check availability	NIST as a whole is a valuable tool. It may be a bit out of date in places, but there is no need to reinvent the wheel if you can help it.
www.aic.nrl.navy.mil/galist/src	Genetic algorithms. Extremely specific.	Public domain	If you already know about genetic algorithms, you probably already know of this source.

Table 3—These are public domain source code repositories. There is good, free code out there. Looking through source code repositories is the easiest way to find what you are looking for.

you're dealing with a fairly specific problem that others have had to solve and isn't overly chip constrained. Unfortunately, it is unlikely to be exactly what is required—a different compiler or a different approach to the problem—but it may be close enough to be worthwhile.

In my search for code, I didn't find much in the way of serial drivers, but I found some nice PID code as well as some filtering code. Most things were pretty specific for the problem they were trying to solve, but there were some good additions to my bag of tricks. My plan is to collect good and interesting code. It's hard at first, but once you have a little, it becomes easier to add on. 📦

Elecia White has been making interesting gizmos for nearly a decade. She has worked on DNA scanners, inertial measurement units, and preschool toys. Elecia is the principal engineer at Logical Elegance, which is a firmware consulting company based in California. You may reach her at elecia@logicalelegance.com.

REFERENCES

- [1] D. K. Rosenberg, "Evaluation of Public Software Licenses," Atlanta Linux Showcase, October 1998.
- [2] "Frequently Asked Questions About the GNU GPL," www.gnu.org/licenses/gpl-faq.html.
- [3] "GNU Lesser General Public License," www.gnu.org/licenses/lgpl.html.
- [4] The Mozilla Organization, "Annotated Mozilla Public License (Version 1.1)," www.mozilla.org/MPL/MPL-1.1-annotated.html.
- [5] B. Perens, "The Open Source Definition", *Open Sources: Voices from the Open Source Revolution*, O'Reilly, 1999, www.oreilly.com/catalog/open-sources/book/toc.html.