Software Freedom
Law Center

1995 Broadway, 17th Floor
New York, NY 10023–5882
tel +1–212–580–0800
fax +1–212–580–0898
www.softwarefreedom.org

A Legal Issues Primer for Open Source and Free Software Projects

Richard Fontana
Bradley M. Kuhn
Eben Moglen
Matthew Norwood
Daniel B. Ravicher
Karen Sandler
James Vasile
Aaron Williamson

Version 1.5.2
4 June 2008

Contents

Foreword

It is a great personal pleasure for me to offer these brief remarks by way of preface to a work composed by my colleagues at the Software Freedom Law Center. As the founding director of the organization, I feel both proud of the lawyers and "laymen" who work at SFLC, and immensely respectful of their achievements. Like every hacker whose late-night solo adventure has attracted a community of programmers doing every year what the originator could not have achieved in a lifetime, I am moved by the experience of becoming merely one among many—another of the minds and hands drawn to the work of inventing a better world. I hope that as you read what follows you too will see some reflection of that basic miracle of sharing, teaching, and growing.

Eben Moglen
New York, NY
February 2008

Chapter 1

Introduction

We at the Software Freedom Law Center are extremely fortunate because we get to provide legal assistance to some of the world's leading free and open source software (FOSS) projects. We are inspired by the hard work and commitment of FOSS developers to produce code that can be freely shared and modified, and it is our mission to help make sure that those developers have a legal environment which allows their work to flourish.

Our intended audience for this Primer is any person interested in a basic understanding of the legal issues that impact FOSS development and distribution. In particular, this Primer, like most of our other public work at SFLC, is addressed to two constituencies. First, we provide creative, productive hackers insight on how to interact with the legal system—insofar as it affects the projects they work on—with a minimum of cost, fuss and risk. Second, we present a starting point for lawyers and risk managers for thinking about the particular, at times counter-intuitive, logic of software freedom. While these are the primary audiences we intend to reach, we hope others will benefit from this Primer as well, and we have purposefully given it a non-lawyer style of communication (for example, by intentionally omitting dense citation of judicial or other legal authority that is the hallmark of lawyers writing for lawyers).

While FOSS development can raise many legal issues, a few topics predominate in our work; these are the issues most integral to FOSS projects. This Primer provides a baseline of knowledge about those areas of the law, intending to support productive conversations between clients and lawyers about specific legal needs. We aim to improve the conversation between lawyer and client, but not to make it unnecessary, because law, like most things in life, very rarely has clear cut answers. Solutions for legal problems must be crafted in light of the particulars of each client's situation. What is best for one client in one situation, may very well not be best for another client in the same situation, or even the same client in the same situation at a later date or in a different place. Law cannot yield attainable certainty because it is dynamic, inconsistent, and incapable of mastery by pure rote memorization. This is why we do not provide forms or other tools for "do it yourself" lawyering, which are almost always insufficient and, in fact, can be very harmful to a project's interests.

The specific topics addressed herein are:

(i) copyrights and licensing,

(ii) organizational structure,

(iii) patents, and

(iv) trademarks.

They are presented in this order because that most closely aligns with the lifecycle of the legal needs of a typical FOSS project. When code is written, copyrights immediately come into being. The terms under which the owner of those copyrights allows others to copy, modify and distribute the code determine whether it is considered "free" and/or "open source." Once a project gains speed, many benefits can be achieved by the creation of an organizational entity for the project that is separate from the project's individual developers. After successful public release of a project, patent and trademark issues may arise that need attention. Thus, this Primer proceeds in what is, to us, a very logical order.

In closing, we are extremely pleased and honored to present this Primer and hope that it will benefit both those FOSS projects we already know and those we look forward to meeting in the future.

Chapter 2

Common Copyright Questions

Many FOSS projects face similar copyright issues. Proper understanding of these issues when the project is young can help avoid problems later. In this chapter, we address some of the common early copyright questions posed by FOSS projects. Because we do not know the specifics of your project, this document provides general information, and not legal advice. If your FOSS project has a specific need for legal advice, please contact the Software Freedom Law Center or seek other legal counsel.

All software is subject to copyright law. The moment you save code to a file, copyright law gives you certain rights to control what other people can do with your work. Because almost everybody who contributes code to a software project has rights with respect to their code, understanding the basics of copyright is essential to running a FOSS project.

A software copyright is the exclusive legal right to control the rules for copying, modifying, and distributing a work of software. A person (or company, foundation, trust, or other legal entity) who has these exclusive legal rights is called a "copyright holder". Legal rules prohibit non-copyright holders from copying, modifying or distributing copyrighted works without permission from the copyright holder.

Copyright holders can permit other people to copy or modify their software. That permission (called a "license") can be as simple as a perpetual, unconditional and universal grant of permission to do any of the acts that are exclusive to the copyright holder.

Other licenses are conditional. They allow people to copy or modify software only if certain conditions are met. If you don't meet the conditions, you don't have permission to copy or modify the software. If you make copies or distribute modified versions of the software without satisfying the conditions (i.e. without permission), you infringe the copyright, which gives the copyright holder access to certain legal remedies. In particular, the copyright holder can sue you for damages or ask a court to order you not to make or distribute further copies.

It is important for projects to understand the conditions in their licenses as well as those in the licenses of code they link to and code they incorporate into their project. Complying with the conditions in the license is essential to avoiding copyright infringement.

If you have general questions about copyright, a good source of information is the U.S. Copyright office's Copyright FAQ.

2.1 Copyleft

One important copyright concept that originated in the world of FOSS licensing is "copyleft". "Copyleft" is a play on the word "copyright". Whereas copyright law has traditionally been used to withhold permission to copy, modify or distribute software, some licenses instead use copyright law to **require** that such permissions be granted.

Copyleft licenses are conditional licenses. One of the conditions you must satisfy before distributing copylefted software is that any changes you make to that software be likewise released under the copylefted license. A copyleft license ensures that all modified versions of your project remain free in the same way. Such licenses are said to keep code "forever free".

FOSS licenses can have stronger, weaker or no copyleft provisions, but they all share a common effect: creation of a large pool of software that can be combined and built upon to create new works. Copyleft licenses require that those who take material from the common pool give something back as well.

2.2 Choosing A FOSS License

There are many FOSS licenses. No one license meets everybody's needs. Circumstances vary because licensing decisions can affect which software libraries you can use as well as the size and character of the community that gathers around a project. Licensing decisions must be made in the context of a project's goals, resources, community and philosophy.

Each FOSS license balances different concerns and attempts in its own way to foster a development community that meets the licensor's needs. For example, strong copyleft licenses like the GPL prioritize ensuring that all downstream recipients receive source code and permission to modify the software. By contrast, the licenses we call "permissive licenses" guarantee the availability of their permissions only for the first generation of the software; they are generally understood to permit recipients to release modified versions under more restrictive terms (including both proprietary and copyleft terms). Some licenses address issues concerning patents,[1] demand acknowledgments of prior authors, or require recipients to take other kinds of actions. Despite their differences, the most widely-used FOSS licenses share a common goal: creation of a large pool of software that can be combined and built upon to create new works.

When starting a new project, you should give particular consideration to the license commonly used by your peers. Many Perl and PHP developers, for example, choose to distribute their code under the same licenses as Perl and PHP themselves. Authors of Linux drivers usually choose GPL version 2. OpenBSD developers often favor the ISC license. By choosing the same license as their peers in related projects, these developers make it easier to share code and create copyright policies that are standard throughout a development community.

Most importantly, when choosing a license for your work, you must carefully consider the licenses of the existing libraries and other code which your work incorporates, adapts, or depends on. Most FOSS licenses place some requirements on the distribution of modified versions or derivative works.[2] Your choice of license for your work may be constrained by the licenses of those existing works. This becomes a practical issue most often when you wish to use a version of the GPL for your work but the license of an upstream work which your work incorporates or depends on may be considered "incompatible" with that version of the GPL.

Aside from these more practical concerns, you will actually adopt more than just terms of distribution when you select a license. A license also signifies a community structure and reflects the developers' own histories and personal convictions. Indeed, a license choice is often the most significant factor in predicting which developers will later be drawn to contribute to the project. Large communities have grown up around specific licenses. Each of these communities

[1] Patent issues for FOSS developers are discussed in detail in § 4 of this document.

[2] FOSS licenses differ explicitly or customarily in how they define the scope of derivative works. For example, GPL licensors usually have an expansive view of what a derivative work is, or assume that the underlying copyright system supplies the appropriate expansive definition. By contrast, the Apache License version 2.0 provides an explicit definition of "derivative work" that is narrower than the definition understood by the typical GPL licensor.

has its own customs, beliefs, and personalities. Sometimes choosing a license means choosing the community that embraces that license.

Finally, we cannot stress enough the importance of picking a license that is already widely used by the FOSS community. A familiar license makes user and developer adoption quicker and easier. Also, if you choose a license already in wide use, there is often substantial licensing knowledge and support available to your projects. Indeed, many licenses are shepherded by specific organizations (e.g., the Apache Foundation's stewardship of their license and the FSF's stewardship of the GPL). Choosing a license with organizational infrastructure behind it can help foster adoption of your software and provide you and your development team with insight into how the licensing models work.

Although it is possible for you to write a new license for your code, it is inadvisable[3]. Each existing FOSS license was crafted to address the specific needs of a particular part of the FOSS community. With so many licenses developed over so many years, it is extremely likely that an existing license meets the needs of your project. If you find yourself feeling the urge to write a new license, it's a good sign that you might need to reconsider your approach.

Below we present several well-known licenses, or categories of licenses, that you might wish to consider for your project, depending on the specifics of your project and its goals. This list is illustrative only, and will be expanded in future editions of this document.

2.2.1 The GNU General Public License

The GNU General Public License, version 2 (GPLv2), is the most widely used FOSS license. Prior to 2007, GPLv2 was the only license in common use that implemented a strong copyleft. In 2007 the FSF published GPL version 3 (GPLv3), the first update of the GPL since 1991. GPLv3 has been adopted by a significant number of projects.[4] The vast majority of FOSS projects license their works under either a version of the GPL or a version of the GNU Lesser General Public License (LGPL), which can be regarded as a more permissive variant of the GPL.

Without knowing the specifics of your project and its goals, it would be impossible to give you a definitive answer as to which license is best for your project.

[3] Often, instead of writing a new license from scratch, an existing license with various *exceptions* can be used. This structure is widely accepted and much clearer than drafting a new license from scratch.

[4] Moreover, most GPLv2-licensed works are licensed under GPLv2 "or any later version", or else do not specify a version number; this is understood at least to modify the copyleft requirement so as to permit modified versions to be distributed under GPLv3 rather than GPLv2.

However, because of its prominence, every project should at least consider a version of the GPL. The GPL enjoys several practical advantages over other licenses and approaches licensing from a different philosophical position as well. Most of these differences and advantages stem from the wide acceptance of the GPL and the fact that it is the only license in common use that implements a strong copyleft.

Because so many projects release under the GPL, there is a large community of code that is licensed under the GPL or at least on terms that permit combining that software with the GPL. If your project wants the benefit of using many existing libraries or you plan to recycle a lot of found code, there's a good chance much of that code will be GPL-licensed. In order to take advantage of that existing code, you'll need to use the GPL or a license that permits combinations with GPL-licensed software.

The popularity of the GPL provides another advantage to projects because a familiar license makes adoption easier. Not only are people generally familiar with the license, but there is a lot of knowledge and support available for projects that use the GPL. The Free Software Foundation (FSF), which wrote and maintains the license, offers some support for developers. They have experience in managing and enforcing GPL copyrights. They publish material that helps people understand how to comply with the GPL. They also operate a compliance lab to help with GPL compliance.

The strong copyleft provision is another key factor. Most developers who get involved in FOSS enjoy the freedom to build on existing work and want to preserve that freedom for others. These developers seek to keep enhancements to their FOSS code free from proprietary restrictions and "forever free". Please see the discussion on copyleft in § 2.1.

There is a large community of developers who care deeply about copyleft, and many of them are more likely to devote their energy to GPL-license projects than non-GPL'd projects. Using the GPL can help in attracting a large development team of existing FOSS authors to a project.

Even if you don't choose the GPL, there are strong arguments for choosing a license that is understood to permit combination with GPL-licensed software (i.e., a license which the FSF, authors of the GPL, call "compatible"). FOSS development flourishes when different codebases can be brought together; licenses that allow such combinations are the most successful. Code licensed under terms that do not permit such combinations are difficult to combine in the same project.

2.2.2 BSD-Style or Permissive Licenses

There are many licenses commonly referred to as "BSD" or "BSD-style" licenses.[5] Most of these licenses differ from each other in only minor ways, which is why they can be grouped together and described generally as "permissive, non-copyleft" licenses.

These licenses are permissive in that they place the bare minimum of restrictions on subsequent development and distribution. Using these licenses is as close to releasing into the public domain as FOSS licenses get. These are not copyleft licenses— they do nothing to preserve free software rights in downstream versions. Moreover, these licenses do not require source code distribution. If you choose a permissive license, other developers can incorporate your permissive-licensed code into their closed-source, proprietary product, and they can effectively conceal the modifications they made to your code.

The advantage of these and similar licenses (i.e. the ISC License) over more restrictive licenses like the GPL is that they are very tolerant of redistribution under a variety of licensing conditions, including under proprietary licenses. For some projects, having their code included in proprietary software is desirable. Many developers believe this may facilitate wide and quick adoption of the technology by *both* proprietary software distributors and FOSS projects.

Other people are troubled by the lack of a copyleft in BSD-style licenses. If you object to the prospect of downstream modifications to your code disappearing inside proprietary projects, consider a license with copyleft provisions.

Here is a "three-clause BSD-style" license:

> Copyright (c) 2008 YOUR NAME HERE
> All rights reserved.
>
> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
> 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer

[5] Other common names include "MIT-style", "two-clause BSD" and "modified BSD" licenses. The word "modified" refers historically to the revision of a more restrictive earlier version of the BSD license. The term "MIT-style" is arguably misleading, since MIT as an institution has used many licenses for its software, including very permissive licenses as well as proprietary ones.

> in the documentation and/or other materials provided with the distribution.
>
> 3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.
>
> THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Here is a sample of the ISC (Internet Software Consortium) license:

> Copyright (c) 2008, YOUR NAME HERE.
>
> Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.
>
> THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Finally, developers who are combining code from permissive-licensed projects with code licensed under GPL or LGPL are encouraged to read *Maintaining*

Permissive-Licensed Files in a GPL-Licensed Project: Guidelines For Developers, published by the SFLC.

2.2.3 The GNU Lesser General Public License

The GNU Lesser General Public License (LGPL) is sometimes classified as a weak copyleft. It is much like the GPL, but it allows distribution of works forming certain types of combinations with proprietary, closed-source software that are generally assumed to be impermissible under the ordinary GPL. In particular, the copyleft requirement in the LGPL does not extend to works that link against your work. The LGPL is sometimes considered a "halfway" point between the GPL (discussed in § 2.2.1) and permissive licenses (discussed in § 2.2.2).

The most widely-used version of the LGPL is version 2.1. In 2007 the FSF accompanied its release of GPLv3 with version 3 of the LGPL, which is explicitly written as a list of exceptional permissions added to GPLv3.

Although the LGPL was originally aimed at libraries, you should not adopt it merely because your project is a code library, nor should you dismiss it because your software is not a library. If enabling proprietary software to combine with your software is important to the goals of your project, the LGPL may make sense regardless of the type of software you are writing.

2.2.4 The GNU Affero General Public License

The GNU Affero General Public License, version 3 (AGPL), is a variant of GPLv3 published by the FSF in late 2007.[6] Like the LGPL, it can be understood as a modified version of the GPL, but whereas the LGPL provides additional permissions allowing you to use code in more circumstances, the Affero GPL contains an additional requirement that must be met if you include Affero-licensed code in your project. This additional requirement involves network deployments of modified versions.

In general, the GPL's source code provision requirements apply only when software is directly distributed in object code form to a user. If you do not distribute the software, you have no obligation to distribute source code. Therefore, if people interact with your software in a way that does not require them to have a copy of it, most of the GPL's source-code provision requirements do not apply.

[6] The AGPL is meant to be the successor to the Affero GPL, version 1, a variant of GPLv2 published by Affero, Inc. in 2002.

The most common example of such interactions is online or networked applications in a client/server environment. For example, when a browser sends a request to a website to perform an operation (such as database entry that occurs by way of a CGI script), the web server performs the operation on the client's behalf. The client does not require a copy of the software to perform the operation.[7]. The GPL imposes no requirement to provide users who visit your website with a copy of the software that runs on your server, even if your server software is GPL-licensed software that you have modified.

In a lot of cases, this makes sense. Servers and clients interact in a lot of circumstances, and offering source code to every client that interacts with every server would be cumbersome. In many instances, it would be impossible.

In some cases, though, offering source code would be trivial, and yet significant improvements are made to server-side software that never get contributed back to the community. For example, because the GPL does not require it, people rarely make the code behind their websites available to users of the site. It is in these cases that the Affero GPL can help.

The AGPL requires that you offer corresponding source code to users who interact with your modified version of AGPL-licensed software over a network. Therefore, if you build a server based on improvements to AGPL-licensed software, those improvements must be made available to all who use the software via the network.

Some copyleft advocates regard the AGPL as the next logical step toward software freedom. As more and more software is delivered as a service instead of as copies of software packages, the Affero GPL can ensure that freedom endures even as distribution dwindles.

The Affero GPL is not right for everybody. Some communities do not want the added burden of packaging their code for release to network clients. The Affero GPL creates code distribution requirements for a class of people the FOSS world has traditionally treated more as end-users than developers. Even though some end-users are now sophisticated enough to customize their installations of open source server software, your project might not want to require them to publish those custom changes. Moreover, there are many individual developers as well as corporate commercial users who believe that AGPL takes the idea of copyleft too far.

If you think the Affero GPL is the license for your project, visit the FSF website

[7] Most modern web applications include many components, including REST or CGI components (which are server-side operations) and client-side operations (via Javascript running in the browser). The latter involves distribution of software in the strict sense. The former is the issue we are focused on here. There may also be circumstances in which distribution of client-side code has copyleft implications for server-side code.

for a copy of the license and some instructions on how to use it.

2.3 Copyright Assignment and Unification

With so many developers generating new copyrights almost constantly as they develop software, most FOSS projects need a process to manage, and perhaps centralize, copyrights.

Some FOSS projects require developers to transfer copyright ownership to the "project" (either by assigning to the founder of the project, or to some legal entity that represents the project) before new code is permitted into the official distribution. Other projects instead leave individuals' copyrights in the hands of each contributor. Still others require contributors to "disclaim their copyright interest" (placing the contribution into the "public domain"), so that the contribution can be included under the main copyright of the work. (The latter of those is usually done for very small contributions.)

These are all valid options, and each has its benefits and drawbacks. The right solution for you will depend on the specifics of your project, and the specifics of each particular contribution. Centralizing copyrights via direct copyright assignment provides some compelling advantages if developers are willing to do so. The mild overhead of receiving such assignments is usually made up for by the legal certainty it gives the software.

In general, the most important reason to contribute copyrights to the project is to enable the project to enforce the license. Unifying ownership of the copyrights gives the project indisputable enforcement power that is both simple and clear. If copyright ownership is scattered throughout a developer community spanning many countries and years, enforcement efforts face additional barriers. With a diluted base of copyright holders, enforcement efforts are hindered by figuring out which pieces were copied, tracking down the developers who contributed those pieces, and then getting them involved in the enforcement action. Especially in cases where it is unclear how much or which code has been copied, the project needs to avoid quibbling about whose copyrights are at stake.

Another reason to unify copyrights is to avoid and prevent later competing copyright claims, such as claims that could be made by employers or developers of proprietary software. A project's copyright assignment process should include questions and instructions designed to identify code with ownership issues, and the code intake process should include mechanisms for dealing with such issues. For example, each contributor should be asked about her employment situation, and whether or not she is (a) working on the software with her employer's equipment or time, and (b) if an agreement exists between the contributor and employer regarding rights to code developed off the employer's premises on the contributor's own time and equipment.

Many contributors may not realize (or remember) that their employment agreement gives their employer a copyright interest in software worked on at home on their own equipment. Asking such questions early as part of a copyright assignment process can expose these problems before the contribution is put into the code repository and becomes a difficult copyright infringement issue for the project.

If your project decides to unify copyrights, developers need not give up their rights to use that code in other projects, even proprietary ones! The copyright assignment can include a license back to the author which grants the author the same benefits as a copyright holder. This includes rights not granted by the project's FOSS licenses, such as permission for proprietary relicensing. The author can thus do anything with the code that she was able to do before she assigned it to the project (except for granting an **exclusive** assignment to somebody else).

2.4 Copyright for Documentation, Websites and Supporting Material

The easiest way to approach licensing issues for documentation and other non-code materials related to your project is to consider them as interrelated and sometimes integrated parts. Because they are often distributed together, it is a good idea to have similar rules apply to both your software and the related materials. Good software should include documentation, and good documentation sometimes includes bits of code.

For the sake of simplicity, many projects release their documentation under the same license as their code. Because of this, we are often asked whether the GPL is a suitable license for documentation. It is, although it is not ideal. The **licensing** questions for documentation and websites are different from those questions in relation to code. Keep in mind that the licenses discussed above were specifically designed for software projects and sometimes have strange and confusing implications when applied to non-software works. We encourage you to investigate carefully the plethora of documentation licenses available in the FOSS world, which include the GNU Free Documentation License and the Creative Commons Attribution-ShareAlike License.

2.5 Copyright Enforcement

Sometimes FOSS licenses are violated. This generally occurs when somebody distributes copylefted FOSS code as part of non-free, closed-source, proprietary products. Often, it is merely because of some confusion about the copyleft terms of distribution (although other forms of violation, such as failure to attribute authors, also occur). If this happens to your project, there are several things you can do about it; but get organized before you act:

2.5.1 Gather the facts

Verify that your copyrighted code is being distributed or copied in violation of your license. Exactly what code is copied in what product, and who holds the copyright to that code? Who, exactly, is violating the license and of what does that violation consist? If you can, use software tools available for analyzing binary files (such as, on GNU/Linux systems, *strings* and *strace*) to confirm that your software is in the product.

2.5.2 Familiarize yourself with the license

Know what licenses apply to the code, who holds the copyrights and what those licenses say. Pay attention to license versions. There are multiple versions and variations of numerous licenses, including the GPL, BSD-style licenses, the PHP license and the Apache license. You should truly and completely understand the obligations imposed by the license before you can explain to a violator how to comply.

The GPL in particular is a lengthy license with a lot of requirements. If you cannot explain the GPL, it will be hard to convince a violator to comply. If you have questions, you can consult the Software Freedom Law Center website and the Free Software Foundation website. These sites have a lot of information that will help you understand the GPL and explain it to others.

2.5.3 Contact other copyright holders

Aggregating claims increases leverage. Groups of copyright holders should designate one point-person to handle communication with the violator. Other members of the group should avoid making statements about the matter (whether to the violator, in public or in private communication) and refer all inquiries to the point-person.

2.5.4 Ask the violator to fix the problem

Be polite but firm. It is often the case that license violations are inadvertent and easily fixed. Offer to help the violator take whatever steps are needed to achieve compliance, and avoid threats of publicity and lawsuits for as long as possible. Make sure the violator understands that your primary concern is the issue of freedom, not a large financial settlement.[8]. Once you convince them of that point, they are likely to respond more positively even if they were initially unresponsive.

The best way to ask the violator to fix the problem is to send a letter, via as many different methods as possible (at least by email, courier service, and fax). Below is some text you might include in a sample letter that can help start a conversation about fixing the problem. Although it specifically mentions the GPL, you should adjust it to the specifics of your license and the violation in question.

> I am writing to you on behalf of the FOO Project. We are the copyright holders of the BAR software package, which we distribute under the terms of the GNU General Public License from http://www.example.org/. Your product, the BAZ application, appears to be a derivative work of BAR.
>
> The only way you could have our permission to distribute copies or derivative works of BAR would be to comply with the terms of our license, the GPL. Unfortunately, it appears that you failed to comply with the terms of the GPL by [BRIEFLY DESCRIBE THE LICENSE VIOLATION HERE].
>
> Most often, such license violations are simple misunderstandings. Please contact me within three days, and we can discuss how best to clear up this misunderstanding.

If polite and friendly requests do not work, contact the Software Freedom Law Center or seek other legal counsel.

[8]However, you shouldn't feel ashamed of asking for a financial component to the settlement, especially if you plan to funnel those funds back into the project. The important thing is to make compliance with the license the primary concern and financial redress as only a secondary concern.

2.6 Copyright Registration

Projects sometimes ask us about registering their copyrights. You do not need to register in order to have a valid copyright. You do not need to register to enforce your copyright. However, registration can provide a project with better options and an increased ability to enforce its copyright license against violators.

If you want to register your copyright, you can download the copyright registration form from the federal Copyright Office as well as instructions on how to fill it out.

Chapter 3

Common Organizational Issues

This chapter addresses the common legal organizational issues facing FOSS projects. It is intended to give these projects background nonprofit corporate and tax information to assist them in the process of evaluating how best to form and manage their organizations and activities. If your free software project has a specific need for legal advice, please contact the Software Freedom Law Center or seek other legal counsel. Social organizational issues, such as how to maintain your project, manage contributions and build community, while very important, are not addressed in this guide.

3.1 Corporate Form

There are a number of different ways to organize a free software project: as an unincorporated organization, as a nonprofit corporation, as part of an umbrella organization or as an individual acting independently, for example. A threshold question to ask is whether or not the project would benefit from adopting a formal organizational structure. Advantages to adopting a corporate form include some protection from liability and having a vehicle for handling assets for the project. A formal corporate entity can also be useful to provide continuity to a project, since the corporate form and identity remains as the individual developers and maintainers increase and decrease their involvement in the project.

Individual Liability

Developers working alone, apart from any corporate form, are not shielded from personal liability for project-related activities. If a developer takes donations or otherwise receives money for working on a solo project, the developer must report that compensation as personal income and pay income taxes on it. Likewise, all legal liability to third parties will fall to the developer. For example, the developer may be liable to users for breach of any express warranties made regarding the software, or any implied warranties that are not effectively disclaimed. Claims by other software producers for copyright, patent, or trademark infringement could also be brought directly against the developer. While there may be advantages to one-developer projects, it is important to realize that the personal assets of the developer could be completely exposed. Formal legal structures, while not a panacea, can help developers to manage their liability and risk.

3.1.1 Unincorporated Associations

Unincorporated associations are groups of individuals acting together as an association without incorporation, and often without undergoing the formalities of formation. Unincorporated associations can be organized with a constitution and bylaws. They generally do not have to register with or report to a state, but will be held liable for federal and state taxes if they do not obtain the appropriate exemptions. In some states, unincorporated associations cannot hold real property, but they usually open a bank account and otherwise hold assets. For unincorporated associations, there is no liability shield against individual liability. The associations can only be sued when the plaintiff has a cause of action against all of its members, but individual members can be held liable for acts they ratify or participate in. While there are some benefits to unincorporated organizations, the rules governing unincorporated organizations vary state to state and often the organizational obligations of unincorporated associations approach that of incorporated ones. This guide primarily discusses the issues raised in forming and running incorporated organizations.

3.1.2 Nonprofit Corporations

Since legally a corporation is usually treated as a "person," the corporation bears the liabilities and receives the benefits of the corporation's actions, rather than the individuals who are involved in the organization's operations. While this is a clear benefit to organizing a corporation, a court could "pierce the corporate veil" and disregard the corporate entity if the individuals involved commingle their funds with those of the corporation or fail to maintain the appropriate corporate records (like minutes of board meetings, for example). If this should occur, the individuals involved would be personally liable for the responsibilities of the corporation. Additionally, individuals could be personally liable for negligent behavior or illegal activities.

Creating and maintaining a corporate form is a lot of work, as we discuss below, and may not be the appropriate organizational structure for a FOSS project. If the project does not consist of more than a few individual developers it may make sense to continue to work in an individual capacity.

3.1.3 Umbrella Organizations and Fiscal Sponsors

Another option available to free software projects is to join an already existing nonprofit organization. There are several tax exempt organizations that act as an umbrella organization and provide fiscal sponsorship to the free software projects that join them. The key advantage to joining an umbrella organization is that new projects do not have to bear the expense or administrative burden of incorporation. Umbrella organizations establish their own board of directors, keep the books for the organization and ensure that the entire organization conducts its activities in accordance with the appropriate state corporate laws and the federal and state tax laws.

Software Freedom Conservancy

Because many of its clients could benefit from the protections of having a legal entity as well as tax exemption status, but were reluctant to pay the fees associated with formation or dedicate the time necessary to start and maintain a tax exempt nonprofit, the Software Freedom Law Center has established The Software Freedom Conservancy. Since its launch in 2006, the Conservancy has grown to include free and open source software projects active in a wide range of fields. Projects that wish to join the Conservancy must apply to be evaluated by the Conservancy's Project Evaluation Committee. Once a project joins, it can receive donations that are deductible to donors under U.S. tax law and benefit from financial and administrative services that the Conservancy offers. The Conservancy has chosen not to charge administrative fees to its member projects and to rely on donations to support its activities at the umbrella level. It does not require that projects assign their copyrights to the Conservancy, nor does it require that they choose any one particular free software license.

Other umbrella organizations

There are several other umbrella organizations organized specifically for free software projects. Each of these organizations has its own rules about joining. Some projects require the assignment of all of the project's copyrights while others do not. Different organizations allow their member projects to have different levels of control over their finances and operations. Examples of such free and open source software organizations are:

- The Apache Foundation, which has a project called Incubator, created to help new projects to join the foundation.
- The Free Software Foundation acts as an umbrella organization for its projects. The FSF's model has a philosophical component and its projects must meet certain requirements prior to joining.
- Software in the Public Interest was originally formed as a corporate structure for Debian and has expanded to include a number of other free software projects. SPI is in the process of developing guidance for new projects interested in joining.

3.2 Incorporation

The first step in establishing an independent nonprofit is to create a legal entity. While it is not necessary to retain a lawyer to assist you with your incorporation, it can be helpful to have the assistance of one to help you through the process and to make sure that the organization complies with all of the relevant state and federal laws. This description of the steps towards corporate formation should be used as only a guide. You should consult with a lawyer and/or examine all state and federal rules closely throughout the formation process.

3.2.1 Where to Incorporate

The process for incorporating varies by state, so you will need to inquire as to the requirements in your home state (or any state in which the organization has offices or is conducting its business). Generally, the incorporator can choose which state should be the nonprofit's home state.

Nonprofits often choose to incorporate in Delaware, where there are fewer administrative hurdles to incorporation and the process is quick (Delaware has expedited same day incorporation for a fee). A potential pitfall, however, is that some states have requirements for corporations formed elsewhere that are conducting activities in the state which could make it more advantageous to keep the incorporation local. For example, corporations organized outside of New York are considered foreign corporations by the state of New York and are required to pay fees and be subject to New York regulations if they conduct a significant amount of their activities or have an office within the state. For this reason, it usually does not make sense for an organization primarily located in New York to incorporate elsewhere. Many other states have requirements that corporations register as foreign corporations doing business in that state which could incur additional fees and administrative burdens to the organization.

In states where the organization does not have a presence in the state that has been chosen for incorporation (for example, having a key director or officer residing in that state), an agent for service of process may need to be appointed. This ensures that there is someone present in the state to receive legal notices and other official documents. There is an entire industry that has formed around this requirement and there are many for-profit companies that can be hired to perform this function. To appoint an agent, a project will generally need to sign a letter of appointment and also pay an annual fee. Many agents will offer other services, such as handling incorporation and annual reporting which may be expensive but can also ease the burden of handling the paperwork directly.

Many states maintain websites that often provide links to forms and other information regarding incorporation, including California, Delaware, and New York.

3.2.2 Choosing a Name

Some states allow organizations to reserve their name in advance of incorporation. If there is some reason for delay in incorporation, using these procedures to reserve a name could be helpful if it is likely that another nonprofit organization or company would be interested in using the same name (See § 5 for a discussion of trademark concerns that may arise in this process). The organization's name should be unique and descriptive. Some states have restrictions on using certain words within the name of the organization. For example, words that imply a certain kind of government office, savings and loan reference or professional license may not be permissible names of the organization without special permission. Examples of this could be "State Police," "Bank," "Lawyer," or "Trust."

Some states also require that the words "corporation" or "incorporated" be used in the name of the organization. In these cases, it will be necessary to include ",Inc" after the name of the nonprofit, even though the corporation is a nonprofit corporation. This might have the result that the name of the organization looks like"FOSS Project Foundation, Inc.", but this requirement will be uniform for states that require it. Some states do not have this requirement and the organization can be named without the corporate indicator.

3.2.3 Formation Documents

There are a number of documents that are necessary to prepare during the formation process. Many existing free software organizations keep this corporate documentation available on their websites and are a good resource to understand the different ways that these documents can be drafted.

A nonprofit's **certificate of incorporation** is the first document that the organization will have to create. The corporate divisions of many states provide forms of their certificate of incorporation on their websites. The certificate of incorporation is a standard document, describing the basic ways in which the nonprofit will be formed. Most nonprofits will organize as non-stock nonprofit corporation, meaning that there are no owners of the corporation. The corporation is created in the public interest so there can be no shareholders or other owners that have a stake in the corporation.

The certificate of incorporation states the project's address and states the purpose of the organization. Generally, it is advantageous to state the nonprofit's purposes as broadly as possible in the certificate of incorporation so that the description encompasses all of the activities that the organization might engage in over the life of the organization. The certificate of incorporation must also include certain provisions if the organization intends to apply for federal tax exemption.

The corporate workings of the organization are set forth in the nonprofit's **bylaws**. The bylaws describe whether or not the nonprofit will have members, how many directors there will be and what their powers are, what officers will be appointed or elected and it can also state financial controls and procedures. The bylaws are adopted by the board of directors, but free software projects have a great deal of flexibility about how that document can be developed. Some projects merely adopt generally applicable bylaws, leaving the directors with significant flexibility in managing the organization. Others create a public commenting process to ensure that the needs of the community are codified in detail in the document.

Meetings of the directors must be recorded in the board's **minutes** that should be kept in the organization's corporate records. Most important decisions about the organization will be decided at these meetings, and the minutes comprise the paperwork support of these decisions. Where it is not possible to organize a meeting time that is sufficient for a quorum of directors, directors can also sign consents in lieu of a meeting (meetings, which generally are required to take place at least once annually, can be by phone or in person; it is not clear in most states whether electronic meetings without any telephony component are sufficient). Consents simply set forth the decision that the directors agree to, and can be signed by each director separately with binding effect on the whole upon signature by the proper number of directors.

Organizations should also apply for an **EIN**, or an employer identification number, from the IRS. As the name suggests, this number is required by federal law if the organization has employees, but it is necessary to have if the organization intends to open a bank account and can be useful to help the nonprofit transact business. An EIN is is analogous to a social security number for corporate entities. The process to obtain an EIN is simple and can be done by phone or online.

3.3 Governance

A project that forms its own independent legal entity must form a board of directors who are responsible for the overall management of the organization. Laws differ from state to state, but many states require a minimum of three directors. Depending on how the project is organized and maintained, a different number of directors might make sense. If the project already has an advisory council or small group of project leaders, those individuals could assume the duties of director. Projects might want to consider including directors that are not directly involved in the development of their software, but have free software project management or nonprofit organizational experience. It can be helpful to have directors with a wide perspective on the board. An odd number of directors can help avoid an even split of votes on contentious issues.

Becoming a director of a nonprofit organization is a serious responsibility and can be quite time consuming. Directors are bound by both a **duty of care** and a **duty of loyalty** to the organization. To satisfy their duty of care, directors must act with the same care that an ordinary, prudent person would exercise in a like position or under similar circumstances. The duty of loyalty requires that directors act in good faith, be faithful to the organization and pursue the organization's best interests. Directors also have an obligation to act in accordance with the mission, rules and policies of the organization.

To satisfy these obligations, directors must stay informed about the activities of the organization so that they can make informed decisions and stay independent. They must promote the organization's mission and avoid acting in self interest. Where potential conflicts of interest arise, directors must disclose the conflicts as soon as possible to the board, avoid influencing other board members and, depending on the nature of the conflict, recuse themselves from voting on that issue. Directors should be particularly vigilant to avoid conflicts relating to contracts and relationships established between the organization and the director as an individual, or with another organization or company in which that board member has influence.

Organizations should consider creating a document setting forth directors' obligations to the organization for potential directors to read and agree to in advance of becoming a director in the organization. This could help avoid confusion over the life of the organization and help the directors to be aware of and remain focused on their duties to the organization.

3.4 Bookkeeping

It is important that nonprofit organizations keep good books and records of their financial activities. This is necessary to be able to present appropriate back-up in the case of an audit or other inquiry later in the organization's lifetime, but also to ease in the preparation of state and federal filings. Additionally, keeping good financial records of the nonprofit's activities assists the organization with continuity and eases any future transition of personnel. For example, in the case where an organization's volunteer treasurer realizes that the obligations of the position are too great and chooses to resign, the transition to a new treasurer is much more likely to be a smooth process if the books are organized and complete, allowing the new treasurer to assume the responsibilities of the position easily. Ideally, financial transactions will be recorded using accounting software, such as GNU Cash. Nonprofits can also employ an external professional bookkeeper to assist with the books. Professional bookkeepers are useful in helping new organizations start running their operations in an organized manner, and can help make it easier to avoid problematic uses of funds such as the commingling of moneys.

3.5 Tax Exemption Recognition

One of the advantages to forming as a nonprofit organization is that the project may be eligible for federal tax exemption as a charitable organization under §501(c)(3) of the Internal Revenue Code. In addition to an exemption from some taxation by the federal and state governments, tax exempt organizations can receive donations for which the donors can take deductions on their own taxes. The IRS has regularly recognized organizations created for the promotion of free and open source software projects as having charitable purposes.

To initiate the process towards recognition of tax exemption, nonprofits must complete the IRS's Form 1023. The form gives basic information about the organization and requires the organization to submit a detailed description of its activities, including the percentage of the organization's activities that are spent on each particular activity. The form also requires that the organization provide three years of financial history or, if the organization has recently been created, a budget of what the organization expects to receive in donations and income and what and how it expects to spend its money. The IRS publishes detailed instructions that provide guidance as to how to properly complete the form. Organizations that receive recognition of tax exemption are obligated to make their applications available on request, which can be a useful resource if another nonprofit that has similar operations has already obtained its recognition.

Nonprofits now have (with extensions) 27 months from formation to achieve tax exemption recognition that applies retroactively to formation. The tax exemption recognition process will take several months, even if the IRS has no objection to the filing. If the IRS has any concerns about the application, the reviewer will contact the organization and provide a list of questions that he or she will give the organization the opportunity to address.

3.5.1 Restricted Activities

Once a nonprofit achieves 501(c)(3) tax exempt status from the IRS, there are certain restrictions preventing the organization from spending money or conducting certain activities. Some specific activities that are restricted (and can jeopardize the organization's tax exempt status) include **lobbying** activities and spending money in ways other than are consistent with the **charitable purposes** of the organization. Also, none of the organization's earnings may inure to any private individual.

Lobbying

A 501(c)(3) nonprofit may not attempt to influence legislation as a substantial part of its activities and it may not participate in any campaign activity for or against political candidates. The law offers no clear test for what amounts to a substantial amount of legislative activities. While one court opinion indicated that lobbying expenditures of less than five percent of an organization's resources did not represent a substantial part of its activities, courts differ in their opinion and others recommend that no more than 15% of an organization's total expenditures be for lobbying. The IRS offers an alternative test called the Expenditure Test which examines certain permissible and impermissible expenditures in an attempt to provide clearer rules for organizations that engage in some lobbying. If your organization is engaging in more than a trivial amount of lobbying activities or resource allocation you should seek legal advice from an attorney.

3.5.2 Public Support Test

The IRS classifies charitable organizations by the ways in which their funding is received. Organizations that are broadly funded by the public are considered to be publicly supported and organizations that received most of their income from a few sources are considered to be private foundations. Private foundations have strict rules and regulations about their use of funds and dealings with contributors that are not applicable to publicly supported organizations. The IRS has established a test to determine whether or not a nonprofit's funding can be considered as "public support." In making the calculation, organizations tally the amount of money received from public donations, where the amount from any one contributor can only be included up to two percent of the total amount of the organization's funding. This amount is then divided by the total amount of funding that the organization received in that year. To be considered publicly supported, organizations must have received over a third of their funds from "public support." There is also a "facts and circumstances" test, applicable where the organization has received more than ten percent of its funds as "public support" and has other circumstances that show the public nature of the organization (for example, the public nature of the board, a strong program to solicit funds from the public or services that appeal to a broad segment of the public).

The IRS acknowledges that new organizations may not be able to meet the public requirements within the first years of their operations and so new nonprofits can apply for an advanced ruling of public support at the same time as they apply for recognition of tax exempt status. Organizations that obtain the advance ruling must satisfy the public support test by the end of the five year period. Organizations that cannot show that they are publicly supported at the end of the period will be required to pay taxes for the previous five year period.

3.5.3 Related and Unrelated Business Income

Nonprofits are also able to receive income from business activities, which can include, for example, selling educational materials, selling t-shirts or other merchandise and selling services. If the business activities are furthering the nonprofit's corporate mission, the income is considered "related business income." Other activities are considered "unrelated business income."

If the income is related business income, the income is not taxed. Related business income is not considered in the public support test, which could be advantageous, since in the calculation of public support, as discussed above (where the numerator is "public support" and the denominator is all of the organization's support), related business income is not included in either of the numerator or the denominator. Related business income is limited in that a publicly supported charity must not depend primarily on gross receipts from related activities.

If the income is unrelated business income, the income will be taxed. Generally, organizations that expect to receive more than $500 in unrelated business income must also pay estimated taxes. In calculating public support, unrelated business income is included in the denominator but not the numerator, so the more unrelated business income that an organization recognizes, the more money it will need to show in public support to maintain its public charity status.

Additionally, if an organization receives most of its support from income, rather than donations, it will move from being a 509(a)(1) organization to a 509(a)(2) organization, in which case it will be subject to an additional test that less than a third of its total support must be from unrelated business activities. For 509(a)(2) organizations, the public support test is a little different for related business income in that it is included in both the numerator and the denominator instead of excluded (there are exceptions to this including an exception that money from persons with certain relationships to the organization be excluded from the numerator).

3.6 Filings

Organizations must remain vigilant about staying compliant with all of the applicable federal and state laws and must be sure to make the appropriate filings on time. Generally, an organization will need to make an annual federal tax filing, an annual state tax filing and often annual corporate filings.

For 501(c)(3) tax exempt organizations, the federal government requires an **Annual Information Return** on Form 990. If the organization receives less than $100,000 a year, it may be eligible to submit a shorter form of this, called a Form 990-EZ. If the organization normally receives less than $25,000 per year, it may not be required to file the Form 990 at all. An organization must file Form 990-T if it has $1,000 or more of gross receipts from unrelated business during the year.

Nonprofits should check to make sure that they satisfy all appropriate state filings. For example, many states require that organizations that receive over a certain amount of income must have an audit performed on their annual financials. New York State, for example, requires that the audit and a copy of the financial statements be filed with the Charities Bureau. If your organization has a significant income, it is recommended that you consult with an accountant and a lawyer to make sure that your organization is compliant with all of the appropriate requirements.

Chapter 4

Patent Defenses for FOSS Developers

A patent is a form of property,[1] giving its owner a monopoly power to exclude others from exploiting an invention[2] "claimed" in the official document through which the patent is issued. In the United States, patents are issued by the U.S. Patent and Trademark Office (USPTO). The document itself is also referred to as "a patent." A recently issued U.S. patent expires 20 years from its earliest effective filing date (typically, though not always, the filing date of the application that matured into the patent). Many patents in force today, in the U.S. and elsewhere, apply to making, using, and distributing software. Any FOSS project therefore faces some risk of assertion of patents against contributors to, and distributors and users of, the project's code. [3]

There is a tendency both to overestimate and underestimate the threat that patents pose to FOSS. Patent holders are unlikely to sue individual developers

[1] Our description of patents as "property" should not be read as legitimization of any particular aspect of patent law. Rather, we are simply observing that patents, as recognized by the applicable legal system, possess characteristics associated generally with forms of personal property. For example, like other forms of personal property, patents can be bought and sold and give the owner the power to exclude others from enjoying or exploiting the legal interest inherent in them.

[2] For simplicity, and following usual legal practice, we use "invention" to mean a *purported* invention claimed in a patent.

[3] In this document we avoid use of the term "software patent," which has no generally agreed-upon definition. Under current U.S. law, software per se is (probably) not patentable, but it is generally a simple exercise in artful legal drafting to represent a software-related invention as a claim covering patentable subject matter (generally by reciting generic, well-known hardware features). Although the details differ, the basic situation is much the same in many other countries, despite a widely-held misconception in the FOSS community that the patentability of software-related inventions is peculiar to U.S. law.

or nonprofit organizations associated with FOSS projects, since they do not hold substantial assets or enjoy a revenue stream from which substantial patent royalties could be extracted. Nevertheless, such lawsuits are not without precedent and might be a tactic employed by proprietary software companies competing unsuccessfully with FOSS projects. Moreover, commercial distributors and users of a FOSS project are a likely target for litigious patent holders, and suits against such parties may adversely affect the entire community surrounding the project, including the developers producing the software. Furthermore, some patent-holding competitors of FOSS projects engage in non-litigious conduct calculated to disrupt such projects and their communities by fostering the perception that the projects carry significant patent risk.

This chapter provides guidance to FOSS developers who wish to protect their projects and their users against patent risk. It focuses on defensive measures that developers can take if they have been threatened by a patent holder with allegations of patent infringement. There are many other issues involving the relationship between patents and FOSS that are beyond the scope of this document.

This document deals solely with U.S. law. Because we do not know the specifics of your project, we provide general information, and not legal advice. If your project has a specific need for legal advice, please contact the Software Freedom Law Center or seek other legal counsel.

4.1 Structure of a Patent

This subsection briefly describes some of the more important features to look for when reviewing a patent. The parts of the patent other than the drawings and claims are referred to as the **specification**.

Every patent has a **title**. The title tends to be worded rather broadly, but it has essentially no legal significance. Some FOSS developers mistakenly assume the title has some bearing on the content or scope of the patent claims; it does not.

Each patent has a seven-digit **number** that identifies it. The front page of the patent also lists the **application number** of the patent, which was used to identify the application before it issued as a patent. The front page also lists the past U.S. patents and other past publications that were formally considered by the patent examiner[4] during the examination or "prosecution" process. (Such publications constitute a form of "prior art.")

[4] Patent examiners are the officials charged with determining whether to issue an applied-for patent.

The **abstract** is supposed to provide a brief summary of the invention disclosed in the patent. As with the title, many non-lawyers make the mistake of assuming that the abstract defines the scope of the patent. In fact, the abstract has little if any legal significance, although it may sometimes have some bearing on the interpretation of the patent claims.

The **drawings** accompany the body of the patent, and are supposed to help illustrate the implementation of the invention described in the body of the patent. Drawings in software-related patents most commonly include system diagrams and flowcharts.

Some patents list **related patent applications** (any of which might already have issued into a patent). There are various ways in which patents might be related to one another. For example, a patent application might have been divided into multiple "divisionals" if the patent examiner determines that the application claimed more than one invention. In other cases, the patent applicant might have filed a "continuation," with additional patent claims, based on the original application. The family trees of patents can be very complex.

The main body of the specification usually includes sections purporting to provide the **background of the invention** and a **summary of the invention**, and always includes a section known as the **detailed description**.

The background of the invention historically tended to give the applicant's view of the prior art, but this has become increasingly uncommon as applicants fear that any statements about the prior art will be used to limit the scope of the patent. Nevertheless, this section may be useful in indicating what is not covered by the patent.

The summary of the invention, despite its name, should not be assumed to give a reliable summary of the invention as claimed in the patent.

The detailed description is supposed to describe a particular implementation of the invention, and is supposed to provide enough information to enable a "person having ordinary skill in the art" to replicate the claimed invention without undue experimentation. Whether the disclosures in most software-related patents actually meet this standard is open to question.

4.1.1 Claims

The **claims**, which are the most important part of the patent, appear at the end of the patent. The claims define what the patent actually covers. When you read a patent, it is usually a good idea to begin at the end, with the claims, and use the rest of the patent primarily as an aid in interpreting the scope of the claims. (It is common for patent attorneys to draft claims before drafting any other part of a patent application.)

Each claim is a single sentence. Claims begin with a "preamble" followed by one or more "limitations." The division between the preamble and the limitations is usually, but not always, clear. The preamble might seem to restrict the claim in some way, such as by stating what might look to you like a field of use. However, you cannot generally assume that the preamble will actually have any limiting effect.

Claims are classified as either independent or dependent claims. Independent claims are those claims that make no reference to other claims in the patent. Dependent claims explicitly incorporate the contents of other claims in the patent. A dependent claim is necessarily narrower in scope than the claim from which it depends, because it includes an additional limitation not present in the incorporated claim.

Software-related patent claims in recently-issued patents often take the form of "system" or "apparatus" claims, "method" claims, and "computer program product" or "computer-readable medium" claims. System claims recite the elements of a system (which might include one or more computers) as a kind of machine or static object. Method claims are algorithmic in form. Computer-readable medium claims typically duplicate the limitations found in corresponding system or method claims in the patent, but are intended to cover software embodied in a storage or distribution medium. Such claims are particularly designed to make distributors directly liable for patent infringement. Computer-readable medium claims are also often used when claiming inventions that focus on data structures and user interfaces.

4.1.2 File Wrapper

Not part of the patent itself, the file history or "file wrapper" of a patent collects all the written communication that took place between the patent applicant and the patent examiner during prosecution of the patent application. It will include "office actions" mailed by the examiner, which may contain rejections of the patent application's claims based on prior art or other grounds, and "amendments" modifying the claims or counter-arguments submitted by the applicant. File wrappers can be ordered from the USPTO through its Public PAIR website or from a commercial service.

Every statement made by the patent applicant that is recorded in the file wrapper potentially has a limiting effect on the scope of the patent. The file wrapper can therefore be valuable when developing non-infringement determinations, discussed below.

4.2 Patent Infringement

To prove that you[5] infringe a patent, the patent holder must show that you make, use, offer to sell, or sell the invention as it is defined in at least one claim of the patent. As those verbs are understood, they overlap closely with the acts that FOSS licenses permit as a matter of copyright law. (For example, "selling" includes any act of distribution, even if such distribution is free of charge.) Informally, and for simplicity, we speak of the software itself as infringing a patent, although strictly speaking this is not correct, and it has probably given rise to misconceptions in the FOSS community regarding whether particular software is "patent-encumbered."[6]

For software to infringe a patent, the software essentially must implement everything recited in one of the patent's claims. It is crucial to recognize that infringement is based directly on the *claims* of the patent, and not on what is stated or described in other parts of the patent document.

If a court determines that you have infringed a patent, the patent holder is entitled to have the court stop you from infringing and assess money damages that you must pay for past infringement. Even if you modify the code to avoid the claims of the patent, you may still be liable for infringement that preceded the modification.

Lack of prior knowledge of a patent is not a defense to patent infringement. It is also no excuse that you independently came up with the ideas implemented in your software. (However, under current law, you may have a defense if you can prove that your independent invention preceded the patent holder's invention, or that your invention preceded the filing date of the patent by at least one year.)

[5] FOSS developers should not get the impression from our use of the pronoun "you" that they are particularly at risk of being sued for patent infringement; as noted above, suits against such developers are unlikely.

[6] Some FOSS developers have assumed that patent law distinguishes between source code and object code, and that liability for patent infringement can be avoided by distributing source code only, but U.S. patent law provides no clear basis for this assumption. Distribution of source code might well give rise to liability for contributory patent infringement, at least, and many claims in issued patents would be literally infringed by the distribution of source code. The First Amendment may place limits on the policing of source code publication by the courts, but this issue has not yet been addressed by the courts in a patent infringement context.

4.3 Becoming Aware of a Patent

There are various ways in which you might become aware of a specific patent. The patent holder might publicize it, a FOSS developer might call attention to it, a user might ask you about it, or you might discover it in a web search. The patent holder might also contact you directly. For example, the patent holder might send a letter accusing you of infringing a patent, or offering to license the patent to you. The guidance given in this document is applicable regardless of how you learn of the patent. If you receive threats from a patent holder, contact the Software Freedom Law Center or another lawyer immediately.

We generally do not recommend that you attempt to search patents or published pending patent applications. U.S. patent law creates disincentives for searching, even though one of the main justifications given for the patent system is that the disclosure in the specification teaches the public how to practice an invention that might otherwise be secret. Those who become aware of the existence of a patent are subject to enhanced damages if they are found to have subsequently infringed "willfully." Moreover, we find that developers often assume that the patents they discover are broader in scope than they actually are, and thus such developers become overly or needlessly worried. If, despite this, you do intend to conduct a patent search, you should seek legal advice first.

4.4 Understanding the Claims

If you have become aware of a patent that you believe may threaten your project, your first step should be to ascertain the meaning of the claims. This is not a simple task, because patent claims are drafted artfully, using jargon unique to patent attorneys, and are often deliberately drafted in an obscure manner. You should first try to determine the "plain meaning" of the claims. Terms in patent claims are presumed to have their ordinary and customary meanings, which include definitions in use in the relevant technical field.

If some of the words in the claims cannot be properly interpreted using the plain meaning approach (for example, if a claim has multiple interpretations), you should interpret the claims based on definitions of terms in the patent specification, the file wrapper, or other sources, such as technical papers or your own technical expertise. Depending on how artfully the patent application was drafted, this may prove to be a difficult task. Patent attorneys sometimes make up their own terminology to obscure the relationship between the claims and the prior art, and they often use non-limiting examples rather than definitions in the specification.

If the patent examiner rejected claims in the patent application over prior art, the file wrapper might show that the prosecuting patent attorney supplied some definition of a term in order to avoid that prior art.

4.5 Building Defenses

Once you have some understanding of the scope of the claims, you should work with your attorney to develop defenses that will reduce the risk presented by the patent. Four important defenses are license, non-infringement, invalidity, and unenforceability. Of these, it is especially useful to develop and document an informed, good-faith argument that (a) even if the patent is valid, your software does not infringe it, and (b) the patent is not valid.

4.5.1 License

You may be able to show that you have a license to use the patent, or some similar immunity from suit by the patent holder. Such promises may take a number of explicit and implicit forms in addition to formal patent license agreements. For example, some patent-holding companies have issued general public pledges or covenants that apply to the use of FOSS. The patent holder might have agreed with a standards body to provide royalty-free access to patents needed to implement the standard. The patent holder might also have granted a license by contributing or distributing code under a FOSS license, as some FOSS licenses include explicit patent license grants. A patent holder providing code under a FOSS license without an explicit patent license grant, such as GPLv2, would nonetheless probably be held to have granted a license implicitly to recipients of the code, though the scope and coverage of such an implied license would be difficult to establish.

Your attorney should carefully examine the document that contains (or implies) the non-assertion promise to determine who and what are actually covered under it. In many cases such promises are quite narrow and will provide no reliable protection for your project.

4.5.2 Noninfringement

A noninfringement determination is a showing that none of the patent claims actually "read on" your software. In other words, your software does not actually implement what is recited in each claim. To make a noninfringement determination, you normally need to show only that there is one limitation of each independent claim that is not practiced by your software.

Because a dependent claim is, by definition, narrower than the claim from which it depends, if your software does not infringe a particular claim, it necessarily cannot infringe any of that claim's dependent claims. This saves you time in making the noninfringement determination, because for each claim that you demonstrate is not infringed, you can ignore the claims that depend from it.

For a claim, such as a method claim, that recites a series of steps, the order in which the steps are listed is generally irrelevant, unless the claim clearly indicates otherwise. Ordinarily, then, if a claim recites a method for doing A, B, and C, you infringe the claim even though you do those steps in reverse order. Moreover, except in unusual cases, it is not a defense to infringement of a claim that your software performs additional steps, or includes additional features, beyond those recited in the claim. Claims typically include verbs like "comprising" or "including" (often linking the preamble to the recited claim limitations), and those verbs are conventionally understood to be open-ended, meaning "including, but not limited to."

The degree to which the patent claim elements must be read literally in determining infringement varies, depending on circumstances in the prosecution history of the patent. In some cases, your software would have to implement each claim element exactly as described in the claim in order to infringe. In other cases, your implementation need only be "substantially equivalent." This depends particularly on whether the patent applicant amended the claims during prosecution in order to get around prior art cited by the examiner. Ask your attorney to examine this issue.

Prepare a Noninfringement Claim Chart

To show that your software does not infringe a patent, it is a good idea to make a "claim chart" for each independent claim. In a noninfringement claim chart, the left column displays each element of a claim, while the right column contains a brief explanation of why your software does not practice that claim element, if that is the case. For a given claim in your claim chart, you need only explain that your software does not practice one of the claim elements to establish that your software does not infringe the entire claim.

4.5.3 Invalidity

Under patent law, in order for a patent to be valid, the claimed invention must have been useful, reducible to practice, novel, and non-obvious to a "person having ordinary skill in the art" at the time that the invention was made. An invalidity defense, therefore, shows that the patent failed to meet one of these requirements. It is most useful for you to focus on determining whether the patent claims were not novel, or would have been obvious even though, in a strict sense, they were novel.

In a litigation context, it is generally more difficult to show that a patent is invalid than to show that you do not infringe it, because an issued patent is presumed to be valid. The presumption of validity places the burden of proof on you as the challenger of the patent. By contrast, patent holders bear the burden of proof of proving that you infringe the patent. However, as we note below, if you request re-examination of the patent by the USPTO, and the request is granted, there is no presumption that the patent is valid during the re-examination proceeding.

In constructing an invalidity defense, the concept of the **priority date** of the patent is important. The priority date is often, but not always, the same date as the application filing date. In some cases the priority date is earlier, as when the patent issued from a divisional or continuation of an earlier-filed patent application, or when the patent is based on the disclosure given in an earlier-filed "provisional application." Ask your attorney to determine the priority date of the patent; it will generally be the earliest filing date mentioned on the first page of the patent.

Non-novelty

To show that a patent claim is not novel, you must produce a single relevant prior art reference that describes ("teaches") **each** element of the claim. While the statutory definition of invalidating prior art is somewhat complicated, in essence the prior art must either predate the priority date of the patent by at least one year, or predate the invention of the patent. Normally you will not know the date of the invention, so, if possible, you should use prior art that was published at least one year before the priority date.

Although other forms of prior art exist, you are best off using earlier patents (and published patent applications) and other printed publications (textbooks, journal articles, conference proceedings, technical reports, software manuals, source code, and so on) that were actually available to the public before the relevant date. If you are looking for material on the web to use as prior art, it is important for you to be able to establish the date of the publication, which is sometimes difficult for material posted online. If you cannot ascertain that the date of a particular publication is on or before the relevant date, you should look for another source of prior art. (It is a good idea for you to establish good dating practices for your own code and documentation in case you or others will wish to use them as prior art in a future challenge or defense to a patent. Use of a revision control system that includes the system date and time on each operation, as Subversion does, is one way to do this.)

If at all possible, you should use prior art references that were not considered by the patent examiner during prosecution of the patent. Patents are presumed valid against any prior art of record. Although the prior art considered by the examiner is generally listed at the beginning of the patent, you should check the file wrapper to be sure that your prior art was not already considered.

Obviousness

To show that a claim is not novel, you must produce a **single** prior art reference that clearly anticipates each element of that claim. If you need to use two or more prior art references to cover all the claim elements, your argument must be that, in light of the prior art, the claimed invention would have been **obvious** to to a person having ordinary skill in the art at the time the invention was made. In essence, you are arguing that the claim is merely an obvious combination of elements that were already well-known in the relevant technical field. It is more difficult to demonstrate invalidity based on obviousness than on lack of novelty.

A recent decision of the United States Supreme Court, *KSR International Co. v. Teleflex, Inc.*,[7] addressed the appropriate standard for determining obviousness, rejecting a rigid formula that had been applied for many years by the Court of Appeals for the Federal Circuit (the appellate court having exclusive jurisdiction over appeals concerning patent law, which is generally seen as having had a pro-patent bias). Although it is too early to know what impact KSR will have on U.S. patent law, it should now be easier to make successful arguments that a patent claim was obvious over combinations of prior art references.

Prepare an Invalidity Claim Chart

As with noninfringement determinations, in constructing an invalidity defense it is helpful to make a claim chart. The first column of the claim chart again contains each claim element. In the second column, for each claim element,

[7] 550 U.S. __, 127 S. Ct. 1727 (2007).

you summarize the prior art that discloses, and therefore invalidates, that claim element. Be specific about the page number or (in the case of patents and published patent applications) column and line numbers of the prior art reference you are using.

Unlike noninfringement analysis, however, to prove a patent entirely invalid, you must show prior art for each element of each claim, including all dependent claims. If you can only show invalidity for some of the claims, you should prepare noninfringement determinations for the rest if you can. Invalidation of one claim will not necessarily help you in invalidating others.

4.5.4 Noninfringement and Invalidity Opinions

In addition to making noninfringement and invalidity determinations, you should obtain a formal opinion concerning noninfringement and invalidity from a patent attorney who is familiar with the patent, its file history, the relevant prior art, and your project. Such opinions of counsel are important under U.S. law because they reduce the likelihood that an accused infringer will be found to have committed willful infringement, which can lead to enhanced damages of up to three times the amount of damages found or assessed.

Ordinarily, damages will not be enhanced if an infringer, having actually learned of the patent, exercised "due care" by investigating the patent and forming a good-faith belief that the patent was invalid or not infringed before engaging in further infringing acts. Obtaining noninfringement and invalidity opinions, and beginning or continuing activity that might infringe a patent in reliance on such opinions, are a way of fulfilling this duty of due care.

4.5.5 Unenforceability

With your attorney's help, you may be able to show that the patent is unenforceable in either a strict sense or a practical sense. For example, the patent might have been invalidated by a court. If the patent is being subjected to a re-examination by the USPTO, the patent holder as a practical matter generally has a more difficult time enforcing the patent while the re-examination is pending.

Moreover, if the patent has expired, it cannot be infringed. While recently-issued patents expire by default 20 years from the filing date, they can expire earlier if the patent holder has failed to pay maintenance fees, which are due every four and a half years. It is estimated that a third of all patents are not maintained beyond the eleventh year of the term. However, even though the patent has expired, if you were infringing the patent before it expired, the patent holder can seek damages for infringements occurring up to six years in the past.

There may be other grounds for unenforceability. For example, the inventors and others closely associated with the patent application might have failed to satisfy their "duty to disclose" to the USPTO all relevant and material prior art of which they were aware.

4.6 Other Measures

In addition to establishing defenses against assertion of the patent, it may be worthwhile to take other measures to minimize your risks from a patent. Two examples are designing around the patent and filing a re-examination.

4.6.1 Designing Around

If it is practical to do so, you may wish to consider ways of "designing around" the patent claims: changing your code to avoid having it fall within the scope of the claims. However, the patent holder will still be able to assert that you infringed the patent prior to implementing the workaround. Designing around may be impractical, however, if it is too difficult to ascertain the precise meaning of the claims, or if the claims are relatively broad. (However, the broader a claim is, the easier it will be to construct an invalidity argument based on prior art.)

4.6.2 Re-examinations

If you have made a strong invalidity determination, you may wish to ask a patent attorney to help you file a patent re-examination request with the USPTO, which potentially results in cancellation of the patent claims, although the filing fees may be out of reach for many nonprofit projects. The USPTO will grant the request and order re-examination of the patent if it considers your request to raise a "substantial new question of patentability" regarding the patent claims. The request must be based on prior art in the form of patents or printed publications; other forms of prior art cannot be used. If the USPTO orders re-examination, prosecution of the patent begins all over again for the patent holder. In a re-examination, the patent claims lose their presumption of validity.

The most common form of re-examination is ex parte re-examination, in which a third party requester whose request is granted has no further opportunity to make submissions in the proceeding, apart from the right to submit a reply if the patent owner files a response to the order granting the request. The current fee for filing a request for ex parte re-examination is $2520. A much more expensive form of re-examination, inter partes re-examination, is also available for more recently-granted patents; the current fee is $8800. It offers the requester opportunities to submit arguments during the course of the re-examination, though it has certain drawbacks in addition to its higher fee.

Re-examinations are usually successful at least in causing the patent holder to narrow the patent claims. It may then become marginally easier for you to make a noninfringement determination for those claims, or to design around those claims. On the other hand, re-examination carries some risk that the patent claims will survive unscathed. In that case, the patent emerges with an even stronger presumption of validity, because it has withstood a challenge based on additional prior art.

4.7 Should FOSS Developers Apply for Patents?

While hostility towards software-related patents is especially prevalent throughout the FOSS community, FOSS developers occasionally ask whether they should apply for their own patents to defend themselves and their projects against patents held by others. Usually this will not be advisable, for several reasons. The costs of applying for and maintaining a competently-drafted patent may be out of reach for an individual developer or nonprofit FOSS project. Moreover, a FOSS project whose developers file for and obtain patents may suffer considerable political damage within the FOSS community, given prevailing community attitudes towards software-related patents. Two other reasons, however, are even more significant.

First, the novelty requirement for patentability is fundamentally in tension with the nature of FOSS development. FOSS development is done in a collaborative, distributed and public fashion. If an invention arises out of such FOSS development activity, the inventors would have to act very quickly to file a patent application, since otherwise relevant public code, documentation and discussions would eventually (or immediately, in most countries outside the U.S.) constitute prior art to the invention, rendering it unpatentable for non-novelty.

Second, merely owning a few patents will not provide an effective defensive counterweight to a hostile competitor that holds hundreds or thousands of patents. It will also be entirely ineffective against so-called patent trolls. Patent trolls are companies that acquire, sue on and license patents but do not produce any products that might infringe others' patents.

Chapter 5

Common Trademark Issues

Like other products, FOSS applications develop reputations over time as users come to associate an application's name with a particular standard of quality or set of features. Trademark law can help protect this relationship of trust and reliance that a project develops with its users; it allows the project to maintain a certain amount of control over the use of its brand. This document is intended to explain how FOSS developers can make effective use of their trademarks. However, because we do not know the specifics of your project, this document provides general information and not legal advice. If your FOSS project has a specific need for legal advice, please contact the Software Freedom Law Center or seek other legal counsel.

5.1 Choosing a Mark

The purpose of a trademark is to identify the source of a product. FOSS projects often use their names or logos to indicate that a particular distribution, module, or upgrade is an official release of the project. In order to make such indications meaningful, trademark law enables a trademark's owner to prevent others from using the mark in ways likely to cause confusion among potential users of the software. Because the law only protects a mark insofar as it serves this identifying function, the most important quality of a good trademark is distinctiveness. A name, logo, or phrase that does not distinguish your program from other products with substantial similarities is unlikely to be afforded trademark protection. For this reason, generic marks consisting of common words that describe the project (e.g., a music player called "Music Player") are generally poor choices.

The "strongest" marks (those afforded the most protection under trademark law) are those which have no other associations, such as a made-up word or an abstract design. Such marks are the least likely to cause consumer confusion[1] because their only meaning is to identify the product. However, these marks might be undesirable from a marketing perspective, even though it is a strong mark from a legal perspective. Often a project will want its mark to say something about the product's purpose. A project need not avoid descriptive marks entirely, but should avoid marks that consist largely of words and images that are generic to the industry or are very similar to those already used to identify similar products.

If two entities are using the same mark on similar goods (such as two software applications), the law favors the one who used it first. Therefore, in stark contrast to patents (see Chapter 4), FOSS developers *should* perform a thorough search before choosing a mark; it behooves a FOSS project to be as certain as possible their chosen mark is not already in use. The best search is one performed by an experienced trademark specialist, largely because she knows where to look. However, resources are also available for those who wish to search on their own. The U.S. Patent and Trademark Office (USPTO) provides the Trademark Electronic Search System (TESS), which allows users to freely search the USPTO's database of registered trademarks. A search of registered trademarks is not enough, however, since U.S. law also provides *some* trademark protection in the absence of registration. Consequently, you should also perform a thorough web search to determine whether anyone is using that unregistered mark.

5.2 Registered v. Unregistered Marks

Though unregistered marks have some legal protections (called "common law" rights), they are limited to those geographical areas where consumers actually identify the trademark with its source. Traditionally, a mark's geographic reach is defined by criteria such as where the product is sold and where it can be found in brick and mortar stores. Such metrics apply poorly to products which are only (or primarily) distributed online, and consequently their geographic reach for the purposes of unregistered trademarks is generally uncertain.

You can minimize this uncertainty by registering your trademark with either a state or the federal government. Registration grants much stronger protections for your trademark if someone else uses the mark in connection with goods similar to the ones described in your registration application. Registration with a state government grants these protections within the state, while registration

[1] "Consumer confusion" is an important test regarding trademark similarity used by the USPTO.

with the federal government grants them throughout the nation.

The protections conferred through registration are not absolute. If someone establishes that they were using the mark before you, their rights to the use of their unregistered mark will still apply. A thorough search prior to registration can help you avoid this situation and any resulting territorial conflict.

For the vast majority of FOSS projects, especially those distributed on the web, federal registration is preferable and sufficient. Though federal registration fees are significantly higher — $275 per mark per goods classification for a federal mark versus an average of $50 for state registration — distribution is rarely or never limited to a particular state. In addition to broader applicability, federally registered trademarks can be enforced in federal courts, and provide a better position from which to register and enforce trademarks internationally.

5.3 The Federal Registration Process

The USPTO's trademark registration process is relatively simple. You do not need a lawyer, and the entire process can be done online (in fact, online registration is strongly preferred; the USPTO charges an additional $100 for paper filing).

Most of the registration requirements are straightforward — a jpeg image of the mark, the date you first used the mark, etc. In addition to this basic information, the application requires that you provide a "specimen" demonstrating your use of the mark in association with a product. For a software product, the specimen can be:

- A picture of the software's physical packaging bearing the mark, if the product has been distributed physically;
- A screenshot of a webpage displaying both the mark and a download link;
- An advertisement for the product; or
- Any other image which demonstrates "the overall context" of how you have used the mark in association with the product.

The specimen requirement only applies if you are currently using the mark. You may also register a trademark based on an intent to use the mark in commerce within six months of registration.

Next, you should make sure that your project fits within one of the descriptions in the USPTO's Acceptable Identification of Goods & Services manual. This manual classifies goods first by a general international goods class, and then by a more specific identification within that class. There are 34 goods classes, but most FOSS projects will be concerned exclusively with class 9, which applies broadly to computing goods. While some projects will relate to other classes (e.g. games, or software intended for use in the medical industry), even for these projects it is generally sufficient to register only in class 9.

If upon searching the identification manual you are not able to find an identification that applies to your project, you must request that an applicable identification be added to the manual before you can register. You may find several applicable identifications within class 9, as many are quite broad. Make note of all of them so as to make your application as comprehensive as possible; though you must pay a registration fee for each class within which you register a mark, you may select several identifications within a class without paying an additional fee.

After you have all of the above information, you should be able to complete the application without any problem. In order to file the (less expensive) online application, you must pay when you apply, either with a credit card or by setting up a deposit account with the USPTO prior to filing.

5.4 Using Your Mark

If a mark is no longer useful to distinguish one source of goods from another, it can lose its trademark protection, regardless of registration. This can happen either through abandonment or genericide. Abandonment occurs if the trademark holder stops using the mark for an extended period of time (usually over three years), or fails to monitor or control how the mark is used. Genericide generally occurs when the trademark comes to be used by consumers to refer to a general class of goods, rather than to the trademark holder's particular product, and the trademark holder does not take sufficient action to prevent such generic use of the mark. By observing the following "best practices" guidelines when using or licensing their marks, most projects should be able to easily avoid abandonment and genericide concerns.

5.4.1 Proper Use of Your Own Mark

When you use your trademark (e.g. on your website, packaging, documentation, advertising, and other materials), provide notice of your claim to the mark. If you have not registered your mark, you may use the "™" symbol to assert your common law rights. If your mark is registered, use the "®" symbol, "Registered, U.S. Patent and Trademark Office," or "Reg. U.S. Pat. & Tm. Off."

Also, use your trademark in a consistent and distinctive manner. Set it apart from other text by using capitalized, italic, boldfaced, colored, or otherwise stylized text. Always use registered marks in the same form which appears in the registration application; do not pluralize singular marks.

Using a project's name in place of its function or general class increases the risk that the name will become generic and thus unprotected. A common example of this is "Aspirin"; formerly a trademark of Bayer, the name was used so frequently as a noun in place of "acetylsalicylic acid" that it became generic in the United States. Because software is functional by nature, there is a particular tendency for users to substitute a program's name for the function it performs (i.e., to use the mark as a verb, as in "grep"). You should avoid using your mark this way.

5.4.2 Others' Use of Your Mark

Trademark holders have the right to prevent the unauthorized use of their marks (or similar imitative designs) when such marks are used in commerce and are used in a manner that is likely to cause confusion or to deceive. Non-commercial use, such as in journalism or literature, is usually permissible. Even in connection with commercial goods and services, it is generally permissible for others to use your trademark in a manner that does not imply an official relationship or sponsorship (e.g., to indicate compatibility), to compare its software with yours, or to indicate that it sells or implements your software.

If you believe that someone is using your mark in a manner protected by your exclusive rights and has not obtained your authorization to do so, you should take action even if you do not object to their use. If you do not object, you should enter into an explicit license agreement with the other party, clearly defining the parameters of their use. It is particularly important to address in the license the quality of product to which your mark may be affixed. You may specify specific quality standards regarding testing, performance, compatibility, and the like. Alternatively, or additionally, you may explicitly retain the right to approve or veto individual uses of the mark. In practice, you should exercise your right to control use of the mark such that the reputation your users associate with your product is effectively upheld by your licensees.

If you disapprove of someone's unlicensed use which infringes upon your exclusive rights, you should send a polite email to the infringer notifying them of your claim to the mark, and that their use is unacceptable. You might suggest a licensing arrangement, contingent on some changes in their usage of the mark. If you believe such an arrangement is impossible, ask the violator to fix the problem. Be polite but firm. It is often the case that license violations are inadvertent and easily fixed. Offer to help the violator take whatever steps are needed to achieve compliance, and avoid threats of publicity and lawsuits for as long as possible. Make sure the violator understands that your primary concern is the project's reputation, not a large financial settlement. Once you convince them of that point, they are likely to respond more positively even if they were initially unresponsive.

As described above, a trademark's legal protections can be lost if it becomes generic in the minds of the public. However, genericide occurs infrequently, and is a greater danger to very famous marks, simply because more widespread use gives rise to a greater possibility of widespread misuse. Furthermore, a trademark owner has no right to enjoin generic use where it is most common — in non-commercial communications amongst the public. Some trademark holders attempt to fight genericide by taking out advertisements clarifying proper use of their mark, but most FOSS projects do not have the finances for such a campaign. Instead, if you are concerned about genericide, you might post guidelines for proper use of your mark on your website. If you notice that someone is using the mark generically, you can point them to these guidelines and politely request that they help you protect your mark by using it properly. However, you have no legal right to stop non-commercial and non-confusing generic use, so demands and legal threats are particularly inappropriate in these communications.

5.5 An example: Project Foo and FooNews

A typical situation for a successful project is that websites spring up to serve the needs of the project's community. This is usually a welcome development, but it does often raise trademark concerns. Here we look at FooNews, a new website devoted to the latest developments in the Project Foo community. Such a website raises a few questions:

- Does the website need permission from the project to call itself "FooNews"?
- In what circumstances would a project want to stop such use of its trademark?
- How does a project give or deny permission to use their mark?

5.5.1 Is permission needed?

Most people, upon visiting a site called FooNews and seeing it report exclusively on news about Project Foo, would be quite likely to assume the site is owned and controlled by Project Foo. It's a reasonable mistake as to the source of the website, and it is precisely this kind of mistake that trademark law aims to prevent. The way the law accomplishes this is to give the trademark holder the power to decide who may use the trademark and who may not. In virtually all cases of the kind described here, the trademark holder does indeed have the power to stop websites from using the mark for the purpose of preventing confusion as to the source of the website.

There are, however, other ways to prevent this confusion in the website's readers. The site could, for example, provide conspicuous disclaimers alerting readers to the fact that the site is operated by entirely different people than the project, that the views expressed and claims made by the site are not those of Project Foo or the people who work on Project Foo. If these disclaimers are conspicuous and definitive enough, that might remove the confusion and allow the site to use the trademark without the project's permission. Still, to avoid any doubts, it's best for both the project and the website to reach an agreement as to what use is permissible.

5.5.2 When to allow use

Way down in the roots of Software Freedom are the same ideas from which the belief in free speech derives. FOSS projects are not often built by people who value censorship, and there is a strong belief within most FOSS communities that projects should thrive on their merits, and not use legal weapons to silence critics and competitors. It is for this reason that in many cases, projects will encourage use of their name even when commercial, proprietary software-producers would forbid it.

The case of Project Foo and FooNews is no different. Although Project Foo can often stop a website like FooNews, it usually has no reason to do so. Calling the site "FooNews" is an excellent way to communicate to readers what the site is about, and Project Foo usually wants people to be able to find information about them. FOSS projects are built by nurturing community, not by using trademark law to crush it.

Sometimes, a project's community can be mean-spirited or critical of the project. Even then, if criticism is offered in good faith, a website like FooNews devoted to pointing out flaws in Project Foo might be useful. In any event, trademark enforcement against people who criticize a project will just engender more criticism. We encourage projects to encourage community-based websites related to the project, even when those websites contain criticism of the project.

The time to become concerned is when people want to use a trademark for purposes that don't further the project's goals. If people trade on your good reputation to sell services that devalue or compete with your project (for example, by implying you endorse their products), it would be foolish *not* to do something about it.

Every project has different goals and every project's community is going to have a different idea of what is acceptable use of the project's name. Whatever action you decide to take with regard to the project's name, make sure your community understands the reasons why you make the decisions you do. If there is a lot of objection to your policies, you might want to examine them and see if there are ways to allow use while still protecting the project's name.

5.6 Trademark Policy

The default trademark rules are sufficient for most Software Freedom projects and most projects do not actually need a trademark policy. Sometimes, though, there is confusion in a project's community of developers and users as to what constitutes legal or acceptable use of the project's name or logo.

Most of the time, a project wants to encourage activity related to the project. When websites spring up that are devoted to supporting a project's users, providing related software and services, and discussing news about the project, that is a good sign that the project is successful and having a wide impact.

Here is a sample trademark policy that Project Foo could use to communicate some simple guidelines to its community about using their name and logo. You can adjust it to suit the needs of your project.

> We at Project Foo love it when people talk about Project Foo, build businesses around Project Foo and produce products that make life better for Project Foo users and developers. We do, however, have a trademark, which we are obliged to protect. The trademark gives us the exclusive right to use the term to promote websites, services, businesses and products. Although those rights are exclusively ours, we are happy to give people permission to use the term under most circumstances.
>
> The following is a general policy that tells you when you can refer to the Project Foo name and logo without need of any specific permission from Project Foo:
>
> First, you must make clear that you are not Project Foo and that you do not represent Project Foo. A simple disclaimer on your home page is an excellent way of doing that.

> Second, you may not incorporate the Project Foo name or logo into the name or logo of your website, product, business or service.
>
> Third, you may use the Project Foo name (but not the Project Foo logo) only in descriptions of your website, product, business or service to provide accurate information to the public about yourself.
>
> Fourth, you may not use the Project Foo graphical logo.
>
> If you would like to use the Project Foo name or logo for any other use, please contact us and we'll discuss a way to make that happen. We don't have strong objections to people using the name for their websites and businesses, but we do need the chance to review such use. Generally, we approve your use if you agree to a few things, mainly: (1) our rights to the Project Foo trademark are valid and superior to yours and (2) you'll take appropriate steps to make sure people don't confuse your website for ours. In other words, it's not a big deal, and a short conversation (usually done via email) should clear everything up in short order.
>
> If you currently have a website that is using the Project Foo name and you have not gotten permission from us, don't panic. Let us know, and we'll work it out, as described above.

5.7 Forking a Project

When a project forks, it may be desirable for the forked project to continue using the original project's marks in some form. In many situations, this is entirely consistent with the purpose of trademarks. If the fork is initiated by existing project members, it is appropriate for the fork to associate itself with the goodwill and reputation of those members who helped develop and promote the original project's marks. Often, common sense or convention supports such use; for example, a fork aimed at creating a mobile version of Project N might call itself "Project N Mobile." On the other hand, the original project may also have legitimate concerns that a fork's use of the mark will cause confusion or unwanted associations in the minds of users.

In order to balance the interests of the projects and minimize conflict, the leadership of the original and forking projects should, before the fork occurs, negotiate an agreement concerning the latter's use of the former's marks. The agreement might provide, for example, that the name of the original project may be incorporated within the fork's name, but may not be used independently to refer to the fork; or that the original project's graphical logo may only be used if it is sufficiently modified so that it clearly refers only to the fork.

Unfortunately, forking does not always occur under circumstances conducive to amicable negotiation. In order to avoid negotiating terms amidst the ill will generated by a "bad breakup," we advise projects to decide on terms outlining acceptable use of trademarks by project forks before a fork is even contemplated, and to place these terms in a membership agreement or policy statement. A membership agreement, of course, would not apply to non-members who seek to fork the project.

5.8 Responding to Cease-And-Desist Letters

Cease-and-desist (or "demand") letters are often the first contact a company will make with someone it believes has infringed its trademark rights. They will typically describe the offending behavior and the applicable laws and penalties, and will demand some action on the part of the recipient (such as removing offending content from a website or product). Often these letters take a threatening tone that can be very intimidating, particularly to recipients who have neither the legal knowledge to evaluate the claims nor the resources to hire a lawyer.

When FOSS developers receive such cease-and-desist letters, it usually is because they have developed some software that replaces an existing proprietary product, and they have chosen a name that closely resembles the mark of that well-known proprietary application. To avoid the likelihood that this will happen to your project, follow the advice in § 5.1 to make your best effort to chose a name that does not resemble existing marks already in use.[2]

If you nevertheless receive a cease-and-desist letter, do not take for granted that its tone reflects the strength of the sender's legal position; the purpose of these letters is generally to inspire compliance with minimal effort, and sometimes scare tactics are effective to this end. Instead, respond reasonably. If the claims in the letter are unclear, ask for more information. Precisely what about your use is objectionable? What statute does the company believe you have violated? If after gathering this information, you are uncertain whether your use is infringing, contact the Software Freedom Law Center or seek other legal counsel.

[2]It turns out that classic FOSS "hacker wordplay" names like "GNU" are actually an excellent way to avoid possible trademark infringement claims. Since "GNU" itself does not resemble the word "Unix" at all, and since when expanded it explicitly tells the reader that the product is **not** Unix (i.e., "Gnu's Not Unix"), a potential trademark holder on the term "Unix" would be hard pressed to make the case that consumers would be confused and think that GNU really is Unix.