13

Open Source Software: Licenses and Leverage

> "Innovation happens elsewhere. Most of the smart people don't work for you."
> —*Bill Joy, cofounder, Sun Microsystems*

A funny thing happened on the way to the twenty-first century: The Internet ended up enabling the free exchange of ideas within communities that span both corporate and national boundaries. Surprising to many, these open communities—especially in software—ended up out-innovating their proprietary counterparts.

Certainly open source software technology *adoption* can ride wild exponentials upward as the technology user community explodes around the typically much smaller nucleus of code contributors. It has also been proven possible to build quite valuable businesses around open source software. Red Hat, for example, has been enormously successful in redistributing the free and open source software components of the Linux kernel and layered applications. MySQL AB, maker of the MySQL open source database, fetched $1 billion from Sun Microsystems in January 2008. Even certain MySQL *binary distributions* are free. (On a typical day more than 50,000 of them are downloaded worldwide!)

The open source model has proven that at times, the best way to control a technology is to give it away. And by "give it away" we mean "release the code under an appropriate open source software license."

In this chapter, we detail how different licenses work and how and why you want to choose one over another. At a basic level, however, all open source licenses work the same. They are just that: software licenses. The license is written in comment lines at the top of a source code file and says, in effect: "You can freely make copies of this code, but only if you follow the rules spelled out here. If you don't follow the rules, then we retract the license and any copy you make will be considered a violation of our copyright." Here are some of the kinds of rules you'll find.

- **Propagation:** Virtually every license requires you to propagate both the license and the original copyright notice.

- **Put-back:** You might be required to republish any changes you made, whether they are optimizations, added features, or bug fixes.

- **Virality:** You might be required to ensure that any "adjacent" code modules carry the same software license, for software you wrote yourself or obtained from other sources.[1]

- **Patent grant:** By using the code you implicitly grant any patents you might have on that code that might infringe to all other users of the code.

- **As–is:** There is no warranty that the code actually works, or isn't somehow infringing on someone's patent. And by you using the code, you won't sue for liability.

- **Noncommercial:** You can use the code for personal, experimental, or academic uses, but you can't sell a product or run your business on it.

At one end of the spectrum are licenses such as BSD and MIT, which are very permissive and mostly just specify propagation. At the other end are very prescriptive licenses, such as many forms of the General Public License (GPL) that are viral and require put-back and patent grants (GPLv3), among other things. The Apache and Mozilla licenses, as well as the Common Development and Distribution License (CDDL), are somewhere in the middle. These are all spelled out in this chapter.

A good mindset is to think of an open source license as a tool. You have some objective in mind and you should use the right tool for the job. Sometimes your goal is to get the widest possible dissemination of your code, in which case you might choose a very permissive license such as BSD. Other times you might want to encourage a lot of downstream sharing and coherence, so you might find the GPL more appropriate.

"Free" Software Licenses

The development and use of free software is overseen by the Free Software Foundation (FSF), established in 1985 to promote computer users' rights to use, study, copy, modify, and redistribute modified computer programs. The FSF supports many free software projects, but particularly the GNU Project and its GNU operating system. The GNU Project was conceived in 1983 as a

way to bring back the cooperative spirit that prevailed in the computing community in earlier days—to make cooperation possible once again by removing the obstacles to cooperation imposed by the owners of proprietary software. The GNU GPL is a backbone of the free software movement.

Note that the word *free* in "free software" pertains to freedom of access and freedom from onerous restrictions, not price. You may or may not pay to get free software. Either way, once you have the software you have three specific freedoms in using it: (1) the freedom to copy the program and give it away to your friends and coworkers; (2) the freedom to change the program as you wish, by having full access to source code; and (3) the freedom to distribute an improved version and thus help to build the community. (If you redistribute GNU software, you may charge a fee for the physical act of transferring a copy, or you may give away copies.)

The following licenses qualify as free software licenses, and are compatible with the GNU GPL (for detailed information and comments about these and other free software options refer to www.gnu.org or http://gplv3.fsf.org/)[2]

- GNU GPL Version 3

- GNU GPL Version 2

- Apache License, Version 2.0

- Artistic License 2.0

- Berkeley Database License (a.k.a. the Sleepycat Software Product License)

- FreeBSD License

- Intel Open Source License

- License of Netscape JavaScript

- OpenLDAP License, Version 2.7

- License of Perl 5, and earlier

- License of Python 2.0.1, 2.1.1, and later versions

- License of Ruby

- X11 License

- License of zlib

The following licenses are free software licenses, but are *not* compatible with the GNU GPL (a module covered by the GPL and a module covered by

the licenses in the following list cannot legally be linked together; in many cases, these licenses have a newer version that may be GPL–compatible):

- Academic Free License, all versions through 3.0

- Affero General Public License Version 1

- Apache License, Version 1.1

- Apache License, Version 1.0

- Apple Public Source License (APSL), Version 2

- Original BSD license

- Common Development and Distribution License

- Common Public License Version 1.0

- Condor Public License

- Eclipse Public License Version 1.0

- IBM Public License, Version 1.0

- Interbase Public License, Version 1.0

- Mozilla Public License (MPL)

- Netizen Open Source License (NOSL), Version 1.0

- Netscape Public License (NPL), versions 1.0 and 1.1

- Nokia Open Source License

- Open Software License, all versions through 3.0

- OpenSSL License

- Phorum License, Version 2.0

- PHP License, Version 3.01

- License of Python 1.6b1 through 2.0 and 2.1

- Q Public License (QPL), Version 1.0

- Sun Industry Standards Source License 1.0

- Sun Public License

- License of xinetd

- Zope Public License Version 1

Nonfree but Free-Sounding Software Licenses

The following licenses *do not qualify* as free software licenses as defined by the FSF. A nonfree license is automatically incompatible with the GNU GPL. Of course, we urge you to avoid using nonfree software licenses, and to avoid nonfree software in general. There is no way we could list all the known nonfree software licenses here; after all, every proprietary software company has its own. We focus here on licenses that are often mistaken for free software licenses but are, in fact, *not* free software licenses:

- Apple Public Source License (APSL), Version 1.x
- Artistic License 1.0
- AT&T Public License
- eCos Public License, Version 1.1
- GPL for Computer Programs of the Public Administration
- Hacktivismo Enhanced-Source Software License Agreement (HESSLA)
- Microsoft's Shared Source CLI, C#, and Jscript License
- NASA Open Source Agreement
- Open Public License
- Reciprocal Public License
- SGI Free Software License B, Version 1.1
- Squeak License
- YaST License

A Closer Look at the GPL

At first glance the number, variety, and requirements of the various licenses appear somewhat daunting. For 95% of you, however, there's a short list of licensing options with which you should become better acquainted: GPLv2 and GPLv3, BSD, Apache, Mozilla and derivatives, CDDL, and Creative Commons. So, allow us to add some contour to the descriptions and comments provided previously, focusing, for now, on the GPL.

Think of the GPL as simply a license that is appended to the source code itself. It says, "Look, you can do anything you want with this code. You're free to make copies of it, to make improvements to it, and to use it commercially in the way you see fit, but you have to play by the rules." And there are some really important rules. One is that you've got to propagate the license. You can't just strip out the code and put it into your own product. Another is that if you fix bugs or make improvements, you have to make those changes freely available. You have to publish that source code and put it back into the community under the same license.

The license can also cover patents that might be on the code. (GPLv3 makes this explicit—there is ambiguity in GPLv2; CDDL is also explicitly granting.) If you play by the license rules, the person who wrote the code gives you a patent grant, saying, in essence, "If I have any patents that would allow me to exclude you from expressing that idea, meaning that they're relevant to that particular source code, I grant you those patent rights." In short, I won't prosecute you as long as you follow the other rules in the license. Now, if you decide not to follow those rules, the patent grant is rescinded, and I can come at you on those grounds. Furthermore, if you use the code you offer a reciprocal grant for any patents you might hold. If you then decide to prosecute users of the code on the basis of your patents, you are also no longer playing by the rules and you lose all rights to the original code. It's a kind of "mutually assured destruction" that is intended to significantly dilute the value of software patents.

The GPL has another aspect to it, as we alluded to earlier—a viral aspect—that is the subject of much debate in the open source community. Basically, it says that if you have a piece of code that is in the GPL, code that it touches (and much debate centers on what is meant by *touches,* but technically it is part of an "aggregate") must also be licensed under the GPL or a permissive license that is compatible with the GPL (e.g., BSD or Apache). People use a lot of techniques to work around this restriction, such as library interfaces and dynamically loadable modules. But the spirit and intent of the license are clear: If you use GPL code, you should be working toward, not against, software freedom.

To be clear: You can't take code under the GPL that you haven't written and place it under another license. And you don't get a patent grant for any of the ideas expressed in that code that you choose to recode under a different license.

It's important that you examine the key differences between GPLv2 and GPLv3, and for that we recommend "A Quick Guide to GPLv3," by Brett Smith, **available at** www.gnu.org/licenses/quick-guide-gplv3.html. Version 3 includes a number of improvements to make the license easier for everyone

to use and understand. It includes stronger protection against patent threats; it offers more ways to provide source code to users; and it is a more globally applicable license. But even with all of these changes, GPLv3 isn't a radical new license; it's an evolution of the previous version. Though a lot of text has changed, much of it simply clarifies what GPLv2 said.

So, whether you choose GPLv2 or GPLv3, by putting something under the GPL you have a lot to say about what can and can't be done with it. Why, then, are there so many other open source licenses, such as Apache, Mozilla, and CDDL? Because the GPL is sometimes considered too restrictive in its terms. A key difference with the Mozilla license, for example, is that it removes the viral requirement and permits code of different licenses to be commingled (assuming that other licenses permit the commingling as well).

This commingling turns out to be really key for situations when you don't control all of the modules in your system or when you want to work off an open base but do some of your own extensions. And yes, that does go against some of the "software freedom everywhere" philosophy implied by the GPL, but it helps to strike a balance between a whole set of competing interests.

Open software is fundamentally about developer freedom. Complementary to developer freedom are developer rights. A code developer (an individual or a corporation) does have rights to the code he developed. It is, after all, the fruit of his labor. By choosing to place that code under an open source license, a developer surrenders some, but not all, of those rights to the community in the hopes of a beneficial exchange: No open source license gives away all rights.

Contributor Agreements

When starting an open source project, the choice of license is intended to be permanent. One question not really considered by open source licenses is how to handle *multiple licenses or relicenses* of the code. There are many reasons why a new or alternative license might be considered. The Linux kernel, for example, is covered under GPLv2, but suppose Linus Torvalds wanted to move it to GPLv3.[3] Linus couldn't do that unilaterally because he is not the sole copyright holder on the kernel code modules. Just like you can't rip someone else's code and put it under a license you choose, you can't relicense it either, unless you get *all of the copyright holders to agree.*

Without some sort of aggregated copyright, every single contributor must be contacted and unanimity reached in order to relicense a code base, or parts of the code must be reimplemented. This is true for all but the most permissively licensed open source projects.

There are other reasons why multiple licenses might want to be maintained. Suppose that a person or company (say, some sort of electronic equipment manufacturer) wants to use your open source project in its device, but it finds the rules of the license too onerous. For example, it might want to make some improvements or additions as points of differentiation, and keep them secret. One option is that the company could *pay you* to give it a version of your code under a proprietary commercial license. That is, you would get compensated for removing the open source license obligations from the code. This Original Equipment Manufacturer (OEM) model is a fairly common one for companies that are trying to build a business around open source software.

A fundamental governance issue for an open community is how the copyright (and other interests) for contributions is maintained.

Contributor agreements (CAs) are used by many companies, open source communities, and other organizations to set forth the terms under which contributions can be made to a project. Sun's CA, for example, is the contractual vehicle for contributions to Sun open source projects such as OpenSolaris, OpenJDK, and GlassFish.

A CA can be a source of clarity about the terms of a relationship—or a source of confusion and heartburn. Here are our answers to frequently asked questions; we hope they'll help you understand the key considerations of signing a CA and how a CA benefits both software engineers and commercial enterprises.

What Does the CA Do?

Typically, when you sign a CA, you agree to share your copyrights with a specific governing body (a company, community, or other organization) and license any patents bearing on your contributions to that organization. You are asserting that your contributions are original works; that you are legally entitled to grant the organization these rights; and that your contributions do not violate anyone else's rights. By accepting a CA, the organization promises that your contributions will remain free and open source software (i.e., the software will be published and will remain available under a free or open source software license). But typically, that is not the *only* license under which the code could be made available. Specifically, a CA does the following.

- It allows an organization to sponsor projects while retaining the ability to offer commercial licenses. Without this ability, a commercial enterprise could not responsibly open-source code bases that in some cases represent hundreds of millions of dollars of shareholder investment.

- It allows the company to protect community members (both developers and users) from hostile intellectual property litigation should the need arise.

- It protects the integrity of a base of code, and in turn it protects the community around that code base: the company, the developer community, and the project's users. In Sun-sponsored projects, for example, Sun acts on the community's behalf as a steward of the code in the event of any legal challenge. This is in keeping with how other code stewards, such as the FSF, defend projects. In order to represent a code base against legal challenges, Sun must have copyright ownership of all the code in that project. Consolidated copyright of code also allows for the possibility of relicensing the whole code base should that become desirable.

- It allows for simpler relicensing. When starting an open source project, the choice of license is intended to be permanent, but the experience of the past few years is that the ability to relicense a project is a useful tool in meeting challenges to free and open source software (and especially challenges from the proprietary software market), and not having that flexibility may be a drawback. Without an aggregated copyright, every single contributor must be contacted and unanimity reached in order to relicense a code base, or parts of the code must be reimplemented. This is true for all but the most permissively licensed open source projects.

- It allows the sponsoring organization to act as a bridge between different communities using the same code under different licenses. This, in turn, allows the sharing of code among open source projects that might otherwise not be possible, and it allows the company to license source code to parties who are not yet prepared to work with an open source license.

The joint copyright assignment also gives the original donor of the code base the ability to offer commercial, binary distributions of the project. Without this ability, it would not be possible for commercial enterprises like Sun to open their technologies.

Do I Lose the Rights to My Contribution by Signing a CA?

No, a CA should ask you only to share your rights. Be wary of a CA that requires you to transfer copyrights to another organization. When you agree

to a Sun Contributor Agreement (SCA), for example, you grant Sun joint ownership in copyright, and a patent license for your contributions. You retain all rights, title, and interest in your contributions and may use them for any purpose you wish. Other than revoking the rights granted to Sun, you still have the freedom to do whatever you want with your code. Sun may exercise all rights that a copyright holder has, as well as the rights you grant in the CA to use any patents you have in your contributions. As the SCA provides for joint copyright ownership, you may exercise the same rights as Sun in your contributions.

Note that a CA does not necessarily give you a say in the relicensing of your contribution and the use of your granted patent rights. Nor can you be certain that your contributions will make their way into the products and distributions that the company actively markets, or that they will be used only for the advancement of free software. However, through the governance processes for each project and community, participants usually have a strong voice in how the code base as a whole evolves. Consult the governance policies of the projects to which you contribute for specific details.

Will I Receive Credit for My Contributions?

This is up to the company that originates the CA. A CA does not obligate the organization to offer any particular form of credit or recognition for contributions; such policies are determined by individual projects. You should consult a specific project's governance and license documentation for more information.

Can I Contribute the Same Works to Other Projects?

Assuming the CA for a project allowed for shared rights, if you contribute to that project you will retain the right to contribute to other projects not sponsored by that organization under any license. Remember, you are asked only to share rights, not to relinquish them. Contribution policies of other projects to which you might want to contribute may restrict your ability to contribute works you've contributed to an organization's project, or to participate in some roles if you have participated in a project at that organization. Consult their policies for more information.

When Should I Sign a CA?

You'll be asked to execute a CA *before* you make any contribution—no matter how large or small your contribution might be. The requirement for a

signed document is an unfortunate consequence of copyright law in some jurisdictions. Here are a couple of guidelines.

- If you'll be contributing on behalf of a company, an officer of your company (usually a VP or higher title) must sign the CA on behalf of the company.

- If you've previously assigned copyright in your prospective contribution to an open source organization (e.g., the FSF) under its contribution policy, you no longer have the ability to assign a joint copyright to a company. However, the open source organization will probably have granted you back an unlimited, sublicensable copyright license to your contribution, and other accepting organizations may also grant back such a license. This kind of grant-back copyright license may allow you in turn to grant to a company all the rights needed under the CA.

You can stop your participation in a project at any time, but you cannot rescind your assignments or grants with respect to prior contributions. This protects the whole community, allowing downstream users of the code base to rely on it. The company originating the CA typically cannot terminate its responsibilities under the CA either.

What if I'm Working for a Company but Contributing as an Individual?

The lines between "employee" and "individual contributor" continue to blur. Let your employer know whether you intend to contribute to an open source project—whether as an employee or as a private individual—and consult the company's IP specialists if there are any potential issues (see also the discussion on employment contracts in Chapter 12).

Software Indemnity

As the popularity of open source software grows, so do concerns that companies and even individual engineers might be risking a lawsuit if they build or run applications that infringe on someone else's IP.

The ghost of the 2003 patent-infringement lawsuit brought by SCO against IBM continues to scare people. SCO alleged that IBM moved technology from UNIX to Linux against the terms of its contract with SCO, violating patents

and trade secrets in the process. SCO sought $5 billion from Big Blue, and also tried to compel Linux-using corporations to license SCO's UNIX.

SCO lost the battle and the war (the company is no longer a going concern and a judge has declared that there is no UNIX in Linux), but the fear remains. Since open source licenses declaim liability for SCO-style attacks, and since most open source software projects aren't exactly flush with cash to pay settlements or lawsuit judgments, open source software is often perceived as riskier for companies than proprietary software would be.

Yes, you can get indemnification for most open source software packages. There are now companies that specialize in this type of "insurance," such as OpenLogic, which provides indemnification coverage for an extensive list of open source packages (as long as you've purchased open source technical support from OpenLogic). Other companies have taken steps to indemnify developers: Red Hat and Canonical both offer their fee-paying customers indemnification for SCO-style threats, and Sun, as we mentioned in Chapter 12, has paid dearly to indemnify the entire Java community against infringement claims made by Kodak.

But the question remains: Is software indemnification really necessary? And does your company need to pursue an indemnification strategy for the software innovations it creates? According to Stephen O'Grady, industry analyst for RedMonk,[4] the risk of prosecution for illegal use of open source software is similar to that of being sued for using proprietary software—and they're both "exceedingly low," from an historical perspective. His opinion is that open source is not uniquely vulnerable to patent claims, and that "no software, open or closed—is risk-free." He adds that he feels it is not a feature for which it is worth paying a premium or altering a buying decision.