American Bar Association Section of Intellectual Property Law

2017 IP West

Open Source Software: Opportunities and Challenges

Panel:

Jonathan Burns, Google
Hanna Kim, Microsoft
David Levine, Red Hat Software
Heather Meeker, O’Melveny
Julie Mowers, Amazon

Moderator: Mark Wittow, K&L Gates

October 11-12, 2017
Long Beach, CA
A Practical Guide to Open Source Software Licensing

Heather Meeker

Second Edition
Open for Business

A Practical Guide to
Open Source Software Licensing

by Heather Meeker

Second Edition
# Table of Contents

Preface.............................................................................................................................................i

**Part I: A Foundation** ..................................................................................................................1
  Chapter 1: The Philosophy of Free and Open Source Software .................................................3
  Chapter 2: A Tutorial on Computer Software ............................................................................11

**Part II: Basic Open Source Theory and Compliance** ...............................................................31
  Chapter 3: Common Open Source Licenses ..............................................................................33
  Chapter 4: License Compatibility ..............................................................................................51
  Chapter 5: Conditional Licensing .............................................................................................61
  Chapter 6: What Is Distribution? ................................................................................................71
  Chapter 7: Notice Requirements ...............................................................................................85

**Part III: Advanced Compliance** ................................................................................................93
  Chapter 8: The GPL 2 Border Dispute .......................................................................................95
  Chapter 9: LGPL 2.1 Compliance ..............................................................................................121
  Chapter 10: GPL 3 and Affero GPL 3 .......................................................................................125
  Chapter 11: Open Source Policies ............................................................................................139
  Chapter 12: Audits and Due Diligence ....................................................................................143

**Part IV: Intersection with Patents and Trademarks** ................................................................151
  Chapter 13: Open Source and Patents (Grants, Defensive Termination) ...............................153
  Chapter 14: Open Source and Patent Litigation Strategy .......................................................179
  Chapter 15: Trademarks ..........................................................................................................183

**Part V: Contributions and Code Releases** .................................................................................187
  Chapter 16: Open Source Releases ..........................................................................................189

**Part VI: Additional Topics** ........................................................................................................201
  Chapter 17: Mergers & Acquisitions and Other Transactions ...............................................203
  Chapter 18: Government Regulation .......................................................................................213
  Chapter 19: Enforcement and Obstacles to Enforcement ......................................................219
  Chapter 20: Open Standards and Open Source ......................................................................253
  Chapter 21: Open Hardware and Data .....................................................................................257
  Forms and Checklists ...............................................................................................................261
  Glossary and Index ....................................................................................................................263
  Case Index .................................................................................................................................269
Chapter 6

What Is Distribution?¹

The conditions of open source licenses—requirements to deliver notices, make source code available, or relicense on the same terms—are triggered by distribution. For almost all open source licenses, if you don’t redistribute the software, you need not meet any conditions to exercise the license. But what is distribution? Twenty years ago, the answer to this question was easy, but it gets more difficult every year.

An American Term of Art

The GPL is in essence a conditional copyright license, and it has no choice-of-law provision. Therefore, theoretically, only an action regulated by the applicable copyright law can trigger application of its copyleft conditions. In the United States, the core commercial right of copyright is called distribution or publication. Therefore, in the United States, the question of what triggers copyleft conditions is identical to the question of what constitutes distribution under copyright law.

GPL 3, which was released in 2007, attempted to internationalize the license to fit with local variations on this concept by using neutral words such as “propagate” and “convey.” Unlike its successor, GPL 2 specifically named distribution as the trigger for copyleft requirements. GPL 2 remains in wide use—and it particularly is the license applicable to the Linux kernel—so the question of what constitutes distribution under GPL 2 is still alive and well in the open source world.

Distribution, though one of the enumerated rights of copyright under US law, is not defined in the Copyright Act (Title 17 of the United States Code). Title 17 grants a copyright owner the exclusive right to “distribute copies … of the copyrighted work to the public by sale or other transfer of ownership, or by

rental, lease, or lending.” The Act states that “offering to distribute copies … to a group of persons for purposes of further distribution, public performance, or public display, constitutes publication.” But this does not define distribution. Where a statute’s terms are ambiguous on their face, the rules of statutory interpretation allow us to look to the statute’s legislative history. The 1976 House Report also does not define distribution but defines publication in the negative by saying, “[A]ny form of dissemination in which a material object does not change hands—performances or displays on television, for example—is not publication.” Later case law equated distribution with publication.

Section 106(3) of the Copyright Act accords to the copyright owner the exclusive right “to distribute copies or phonorecords of the copyrighted work to the public by sale or other transfer of ownership, or by rental, lease, or lending.” Put differently, the copyright owner has the exclusive right to publicly sell, give away, rent, or lend any material embodiment of his work. As the legislative history of this section shows, the definition of distribution is “virtually identical with that in the definition of publication in section 101.” Thus, in essence, exclusive right of distribution is a right to control the work’s publication.

Determining When Distribution Has Occurred

In the United States, therefore, distribution means providing a tangible copy to another person. The question of what constitutes distribution therefore devolves to two questions: What is a tangible copy, and what is another person?

The transfer of the work must be made “to the public” to trigger the definition of distribution under the Copyright Act. In the absence of a statutory definition of the phrase to the public, courts have held that a “limited” distribution that “communicates the contents of a manuscript to a definitely selected group and for

---

2 17 U.S.C. Section 106(3).
a limited purpose, and without the right of diffusion, reproduction, distribution or sale,” is not distribution to the public.\(^9\)

In other words, a distribution is a “general” publication if it is not made (1) to a limited group; (2) for a limited purpose; and (3) “without the right of diffusion, reproduction, distribution or sale.” The legislative history of the Copyright Act makes it clear that, “when copies or phonorecords are offered to a group of wholesalers, broadcasters, motion picture theaters, etc., publication takes place if the purpose is further distribution, public performance, or public display.”\(^10\) Thus, even if the work is distributed to a single person or entity, the publication would be general if the recipient is free to diffuse, reproduce, distribute, or sell copies of the work.

In the contemporary world of information technology, many activities stray close enough to the transfer of a copy to challenge the boundaries of this definition of distribution. It is these activities that make the question of what is distribution under GPL of such great interest to companies implementing day-to-day strategies for GPL compliance.

Starting at the baseline, the most obvious business case is that of a distributed product. Whether the product is software alone or a hardware product as well, businesspeople understand what it means to sell a product and for it to change hands. Companies trying to comply with open source licenses like GPL 2 therefore have more difficulty assessing activities that they do not consider to be the business case of commercial distribution but that may nevertheless constitute distribution under the law. This chapter discusses those other business cases, from the clearest to the murkiest, as a matter of law.

**A Clear Case in the Clouds**

Companies often wonder whether software transmissions or remote use—sometimes called the ASP or SAAS model, or cloud computing—constitute distribution.

While this is one of the most controversial aspects of free software licensing, it is not a difficult interpretation question under US law for GPL 2. Advocates of

---


free software have long recognized that if the trigger for copyleft requirements is distribution, increasingly popular cloud computing models will circumvent those requirements. This is sometimes referred to as the “ASP loophole.”\(^{11}\)

During the drafting of GPL version 3, this issue engendered significant controversy. At one point, a variation on GPL 3 was proposed to allow the author to select an option that would cause online use to trigger copyleft requirements. Ultimately, this variation was removed from GPL 3 and memorialized in an alternative form of the license known as the “Affero GPL.” The basic form of GPL 3 makes it clear that ASP or SAAS use does not trigger copyleft requirements. In GPL 3, copyleft is triggered by *conveying* rather than by distribution, and “[t]o ‘convey’ a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.”\(^{12}\)

Under US law, distribution requires actual transfer of a copy, in whatever form. Therefore, under US law, SAAS use—which involves accessing software without transfer of a local copy to the user—does not trigger copyleft requirements under GPL.\(^{13}\)

**The Edge Cases**

Leaving aside the two relatively clear business cases of a distributed product (which clearly constitutes distribution) and pure SAAS deployment (which does not), we turn to some of the edge cases that also are common business activities but do not fall so neatly on one side of the distribution coin or the other.

- **Employees.** While companies often worry about this case, it is not a difficult one. Clients often ask whether “internal distribution” within a corporation triggers copyleft requirements. However, under law, there is no such thing as internal distribution because corporations and their employees are considered a single legal person. Therefore, one employee’s providing a copy of software to another employee of the same company in the course of performing their duties as employees is clearly not distribution; while

\(^{11}\) The term is often attributed to Richard Stallman, but that may not be accurate. See the April 3, 2007, interview with Mr. Stallman in Groklaw, in which he says the term is misleading: www.groklaw.net/articlebasic.php?story=20070403114157109.


\(^{13}\) It is worth considering that even in SAAS implementations, some components may be distributed. Today, most SAAS is accomplished only via a browser, so client software is no longer a common requirement to use SAAS. However, there are always exceptions, mostly notably JavaScript or mobile applications. Keep in mind that these are usually clearly distributed and would be subject to copyleft requirements.
it may be a transfer of a copy, it is not a transfer to another person. Free software advocates sometimes refer to this as providing *private copies*.

- **Independent contractors—individuals.** Companies often engage individuals as independent contractors rather than as employees. Emerging growth companies, in particular, do this to avoid the regulatory overhead costs (such as employment taxes) associated with hiring employees. The function of a contractor in such cases is nearly identical to that of an employee; however, because a contractor is not an employee, providing a copy of software to a contractor could be considered distribution. This is one of the thornier areas of GPL 2 interpretation, and it is discussed in more detail below.

- **Independent contractors—consulting firms.** Companies often hire small consulting firms to develop, test, or support software. These consultant entities often consist of a few persons working in a team, but their functional relationship to the company is similar to that of an individual consultant or an employee. Individuals in small consulting firms are not, legally speaking, employees of the company, and therefore, providing a copy to them is probably distribution. However, there may be arguments that the copies are not intended for public availability and that, thus, transferring them is not publication and therefore not distribution. This argument has risks, but it is probably supportable under law, particularly if buttressed by a written consulting agreement that recites the parties’ intentions. This business case is very similar, in a legal sense, to the engagement of an individual contractor.

- **Independent contractors—outsourcing.** Larger companies often outsource entire business areas such as software development or software support. Outsourcers are clearly separate companies, rather than employees, so providing a copy to them is clearly providing a copy to a person other than the company. However, some outsourcing companies provide “leased” staff to work on servers and equipment owned or controlled by their customers. In this case, IT companies may reasonably make arguments that copies made available to those persons have not been transferred outside the companies’ control. This argument may be less successful, however, for outsourcing companies who are outside the United States—as most are. The international divide may make it unclear which body of law will determine what is distribution under GPL.

- **Cloud providers.** With the rise of cloud-based computing services such as Amazon Web Services, companies worry about whether uploading soft-
ware to a cloud service is distribution. Although the question is subject to some debate, the answer is probably not. Although technically this involves the transfer of a copy to the cloud service provider, the virtual space in which the copy resides is under the control of the company user; cloud services agreements generally do not provide the cloud service provider with any control over the information stored in a cloud account. Any transfer of a copy, then, is not “general” under copyright law.

- **Subsidiaries and affiliates.** Companies often create affiliate structures to conduct business for various strategic reasons such as tax planning, needing to do business in other countries through local entities, or engaging in a particular line of business. For example, a company may use a copy of the Linux kernel, which it has modified for its own purposes, to run an online service. It may provide this modified kernel to a subsidiary or affiliate in Europe or China to offer a local service. For tax, regulatory, or other reasons, it may be important to locate the servers for the business in Europe or China in those territories. If the recipient entity is a wholly owned subsidiary of the company, the company has a good argument that, due to unity of ownership, the copy is a “private copy” that has only been given to the company itself and that, therefore, no distribution has taken place. This argument is also reasonably strong for a majority-owned affiliate because the parent effectively exercises control over the affiliate. But if the recipient is a minority-owned affiliate, the company faces a more serious concern over whether distribution has taken place. This scenario is quite common, particularly where companies have little choice but to create minority-owned operating entities in territories, like India or China, that impose significant restrictions on foreign ownership of businesses operating within their borders.

- **Mergers and acquisitions.** US law can be quirky and counterintuitive on the subject of assignments by operation of law in connection with mergers and acquisitions. An assignment of a contract (or a license) occurs when one party to the contract transfers its rights to another. Therefore, for instance, if a corporation enters into an agreement with another party, it may be able to transfer that agreement to another corporation—depending on what the agreement has to say about it. Contracts are generally considered

---

14 In addition, because the copyleft requirements of GPL only allow binary recipients to seek source code copies, where the recipient is a majority-owned affiliate, the issue may be moot; the recipient would simply never make the request.
assignable under US law, but intellectual property licenses are subject to different rules. Generally, nonexclusive copyright and patent licenses are not assignable. Therefore, if a corporation takes a nonexclusive license to a patent, it cannot transfer the license to another corporation unless the license agreement expressly allows transfer. To make matters even more complicated, there are courts that have held that an acquisition—even a transaction such as a reverse triangular merger in which the target entity survives—can be an assignment by operation of law. Even if the licensee is the same corporation before and after the acquisition, the license may not be exercisable after the transaction. This rule of law may also have implications for the definition of distribution. If a change of control is an assignment by operation of law, one might logically conclude that it also constitutes providing a copy to another entity and thus a distribution triggering copyleft obligations. Keep in mind, also, that the effectuation of some forms of M&A transactions, such as asset sales, are clearly assignments and also likely to constitute distribution under GPL 2.

- **Productization.** Although this business case is not complex from a legal standpoint, it is such a frequent trap for companies managing open source compliance that it is worth mentioning in any discussion of distribution issues. Companies that offer SaaS solutions tend to rely on the fact that they are not distributing their products to ensure their GPL compliance. They do this by merely avoiding licenses like Affero GPL that have requirements even in the absence of distribution. However, this can be a dangerous strategy. For a business development manager who is not focused on legal and technical niceties, it is easy to cause transactions to trip over the distribution line. A company with a SaaS offering may, for instance, approach a customer operating in a highly regulated market (such as health care or financial services) that will insist that the SaaS offering be operated via a private instance on the customer’s premises or on servers under the customer’s control. This demand usually arises from security or regulatory auditing concerns. From the business point of view, a private instance of a SaaS product is a technical detail. But of course, providing a copy to the customer will likely constitute distribution. If the company’s

---

15 Other than special kinds of contracts, where assignment would change the basic nature of the contract, like contracts for personal services or requirements contracts. See Restatement (Second) Contracts, Section 317.

16 For patent, see *PPG Indus. Inv. v. Guardian Indus. Corp.*, 597 F.2d 1090 (6th Cir. 1979). For copyright, although the law is conflicting see, e.g., *SQL Solutions, Inc. v. Oracle Corp.*, 1991 U.S. Dist. LEXIS 21097 (N.D. Cal. 1991). This is an unpublished decision and arguably contrary to the California Supreme Court’s view in *Trubowich v. Riverbank Canning Co.*, 182 P.2d 182 (Cal. 1947).
open source compliance strategy hinges on refraining from distribution within the context of a SAAS model, the company may find that it cannot deliver a compliant product in any reasonable amount of time—usually because it has intermixed GPL- and non-GPL-compatible code or has not properly kept track of open source elements in the product.

With these edge cases in mind, we now turn to extrinsic evidence of the meaning of GPL 2 and best practices in managing distribution issues.

**The FSF View**

The GPL 2 FAQ, promulgated by the Free Software Foundation (FSF), offers the FSF’s insight into what it considers a distribution that would trigger copyleft requirements. For example, one of the FAQs is as follows:

> Is making and using multiple copies within one organization or company “distribution”?

> No, in that case the organization is just making the copies for itself. As a consequence, a company or other organization can develop a modified version and install that version through its own facilities, without giving the staff permission to release that modified version to outsiders.

> However, when the organization transfers copies to other organizations or individuals, that is distribution. In particular, providing copies to contractors for use off-site is distribution [emphasis added].

The FAQ also discusses a transfer between an organization and a majority-owned subsidiary:

> Does moving a copy to a majority-owned, and controlled, subsidiary constitute distribution?

> Whether moving a copy to or from this subsidiary constitutes “distribution” is a matter to be decided in each case under the copyright law of the appropriate jurisdiction. The GPL does not and cannot override local laws. US copyright law is not entirely clear on the point, but appears not to consider this distribution [emphasis added].

17 [www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#InternalDistribution](http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#InternalDistribution). This same FAQ appears in the GPLv3 FAQ as well ([www.gnu.org/licenses/gpl-faq.html#InternalDistribution](http://www.gnu.org/licenses/gpl-faq.html#InternalDistribution)).
If, in some country, this is considered distribution, and the subsidiary must receive the right to redistribute the program, that will not make a practical difference. The subsidiary is controlled by the parent company; rights or no rights, it won't redistribute the program unless the parent company decides to do so.\footnote{www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#DistributeSubsidiary. This same FAQ appears in the GPLv3 FAQ as well (www.gnu.org/licenses/gpl-faq.html#DistributeSubsidiary).}

In this FAQ, the FSF acknowledges that, at least in the United States, a transfer to or from a majority-owned and majority-controlled subsidiary may not constitute distribution. Further, the FSF gives weight to one organization’s effective control over another to determine whether the two entities are effectively one entity for the purposes of the analysis.

There is also discussion in the GPL 2 FAQ about providing modifications of GPL code under a nondisclosure agreement:

Does the GPL allow me to develop a modified version under a nondisclosure agreement?

Yes. For instance, you can accept a contract to develop changes and agree not to release your changes until the client says OK. This is permitted because in this case no GPL-covered code is being distributed under an NDA.

You can also release your changes to the client under the GPL, but agree not to release them to anyone else unless the client says OK. In this case, too, no GPL-covered code is being distributed under an NDA, or under any additional restrictions.

The GPL would give the client the right to redistribute your version. In this scenario, the client will probably choose not to exercise that right, but does have the right.\footnote{www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#DevelopChangesUnderNDA. This same FAQ appears in the GPLv3 FAQ as well (www.gnu.org/licenses/gpl-faq.html#DevelopChangesUnderNDA).}

Many companies find the distribution question confusing because they find this FAQ confusing. In this FAQ, the FSF considers two different scenarios: (1) the contractor releases the modified code to the public generally at the direction of the client, and (2) the contractor releases the modified code to the client under the GPL, and the contractor promises not to release the modified code to anyone else. Unfortunately, this FAQ section does not specify whether “a modi-
fied version” refers to a modification of the contractor’s own GPL code, to GPL code that may have been already modified by the client, or to a modification of third-party code. Clearly, these three situations could be analyzed differently. If the FAQ refers to GPL code owned by either the client or the contractor, it is a trivial question; obviously, the owner of GPL code can choose to deliver that code under GPL terms or not as it sees fit, because an author (as licensor) is not bound by the copyleft obligations of GPL—only the licensee is. If the FAQ refers to modifications to third-party code, it implies that even if the delivery of the original code constitutes a distribution, that distribution does not trigger the copyleft obligations of GPL.

Other information promulgated by the FSF suggests that this FAQ element is not intended to address third-party code. But that is, by far, the most common situation: a company wants to use some GPL code but needs modifications, so it finds an expert in the code willing to modify it on a contract basis. In fact, this scenario is so common that it is touted as one of the advantages of open source software. But the company may not plan to ever distribute the software. Therefore, if providing the code to the consultant is distribution that triggers copyleft requirements, the company will likely be unwilling to engage the consultant.

The FSF’s view is problematic for a couple of reasons. First, a practical problem: companies that hire consultants simply don’t distinguish between the business cases of in-house and contractor development. They do not expect to encounter a completely different GPL compliance landscape based on the distinction. Because FSF’s view contravenes business expectations, it is a trap for the unwary. Second, a legal problem: the provision of code for development purposes is more akin to “communicat[ing] the contents of a manuscript to a definitely selected group and for a limited purpose, and without the right of diffusion, reproduction, distribution or sale” (i.e., not publication under copyright law) than it is to common notions of redistribution or publication. Therefore, there is a strong argument that such a transfer is not distribution under the law.

The International View

It is important to keep in mind that the distribution question as it is analyzed here is largely unique to US law. Because GPL 2 does not have a choice-of-law provision and is a conditional copyright license, it governs only what is protected via local copyright law. A full discussion of the tenets of international copyright law bearing upon this issue is beyond the scope of this chapter, but it seems likely that the question would have different answers outside the United States. The Berne Convention for the Protection of Literary and Artistic Works, as amplified by the WIPO Copyright Treaty, provides for a right to “make avail-
able” a literary work. This may be broader than the US notion of distribution, and most importantly, it could include SAAS offerings. Therefore, the triggers for copyleft obligations based on activity outside the United States may have a lower threshold than those based on activity in the United States.

**Best Practices for Contract Drafting and Deal Structuring**

As lawyers in private practice await clarity in the common law on distribution issues, they may wish to consider implementing certain drafting and structuring practices to clarify their clients’ intent or to minimize the uncertainty of the results should the courts later announce decisions on distribution questions. Such practices cannot address all distribution issues should there be a contrary court pronouncement, but they might help discourage claims, provide evidence of intent, or reduce confusion when those not directly involved in the deal are later asked to assess distribution issues.

**Development Agreements**

To avoid confusion about whether development activities constitute distribution, consider adding language such as the following to your development agreements:

- “Contractor shall conduct development services only on systems and equipment under the control of Customer.” This language limits work to customer-controlled servers. This will address whether a distribution has occurred, the theory being that even though the contractor is a separate person, no copy of the software has been transferred.
- “Contractor acknowledges that it is performing the development services solely for the benefit of Customer, and solely as directed by Customer, and shall not make any copy of the Software available to any other person or entity.” This language states that copies are intended to be private, thus addressing the situation that the FSF FAQ says constitutes distribution.
These approaches are attractive because they comport with customary confidentiality provisions and “work made for hire” provisions in development agreements, which often recite customer control of the development activities to support treatment of the work as “work made for hire” under *CCNV v. Reid.*

**Mergers and Acquisitions**

- Avoid delivery of GPL software. Particularly in asset purchase deals, determine whether there is a reasonable way to refrain from delivery of open source packages in favor of having the buyer download them directly from the original source or a third-party source. This approach is useful mostly in situations in which drivers or other significant original code belonging to the seller is being delivered. It is not useful when the seller is delivering integrated modifications. In that case, the seller would deliver only its additions, and the buyer would receive third-party open source code separately. Clearly, if third-party open source code is extensively modified, this strategy may not be feasible because it would be so difficult to separate the seller’s code from the third party’s code. However, companies that are very conservative on this issue may deliver only diffs, or patches, in an attempt to avoid delivery of any third-party GPL code. Keep in mind that distribution is usually an issue for the seller, not the buyer. Therefore, asset purchases that consist of all the assets of the seller entity may render the concern moot, but a seller’s divestiture of partial assets, business lines, or product lines may cause the seller to have concerns about GPL distribution. A seller wishing to sell its own code may find buyers unwilling to pay for that code if the code must be delivered under GPL.

**SAAS Agreements**

- Avoid drafting that confuses SAAS with distribution. There is some controversy among technology lawyers as to whether SAAS agreements are licenses or merely service agreements. Sometimes, as an artifact of their

---

20 *Community for Creative Non-Violence v. Reid*, 490 U.S. 730 (1989), held that the factors for determining whether a work of authorship is a work made for hire (owned by the company) or not (owned not by the company but by the author) are, among others, the level of skill required to create the work, the source of the tools used in creating the work, where the work was created, the duration of the relationship between customer and author, the extent of the contractor’s discretion over when and how long to work, and whether the work is part of the regular business of the customer or consultant. Therefore, many consulting agreements recite where work will be performed, as well as other facts that might bear on whether distribution has occurred.
business antecedents in distributed software, SAAS agreements are drafted so much like distributed software licenses that it is difficult to tell the two apart. Although the distribution question would likely turn primarily on the supplier’s actions, not merely on document drafting, it is best not to hurt your position by using a SAAS agreement that reads like it covers a distributed product.

**Intercompany Agreements**

- Recite intent not to distribute. In software agreements between corporate affiliates, parent entities may wish to clarify that no distribution is intended, much in the same way as recommended above for consulting or development agreements. This may seem obvious, but in fact, intercompany technology licenses are often not drafted by technology lawyers. Instead, they are drafted by tax lawyers or corporate lawyers who are documenting intercompany arrangements for the purpose of managing imputed tax issues, rather than precisely considering intellectual property issues. It is crucial to review these agreements with a view to open source as well as intellectual property issues.

**An Enduring Puzzle**

It is unlikely that the federal courts in the United States will answer these distribution questions anytime soon. The open source enforcement actions that have been brought to date have not resolved these questions. Given that other heady issues (such as the scope of derivative works under GPL 2 and the interaction of patent law and open source licensing) are still unclear in open source law, they may not be ripe for dispute. Also, most authors who release code under GPL 2 are simply not focused on issues like intercompany agreements and mergers or acquisitions. This is because they are primarily technologists rather than corporate strategists. If GPL authors generally do not intend to enforce their rights in these edge cases, there may not be a constituency that is interested in bringing a lawsuit that will make law in this area. It therefore seems likely these questions will persist as long as GPL 2 remains a widely used license, and based on the prevalence of the Linux kernel alone, this will be a long time. Companies assessing open source compliance should be sure they have identified the types of distribution that are most likely to be questioned so they can use open source software with confidence and plan their transactions in a way that comports with their open source compliance strategy.
Today, it is considered a best practice for a company to have a written open source policy. In some areas of law (such as employment law), the fact of having a written policy has a direct bearing on liability, but for open source, the policy is entirely functional. Copyright infringement is largely a strict liability regime—meaning that intent is not an element of infringement. Therefore, a company will not be able to avoid liability for the infringing acts of its employees merely because it has a written policy. A written policy that everyone disregards is worthless and, in fact, can be counterproductive because it sends a signal that the company will look the other way when violations occur.

Written policies need not be long or complicated. But having them in writing is useful, particularly when a company has multiple development teams who may be in different countries and may speak different languages. Having a written policy gives these teams the time and opportunity to read and understand the policy, ask questions, and refer to the policy when necessary.

However, there is no one-size-fits-all policy. Here are some things to consider if you are contemplating an open source policy for your business.

**Start Small**

Your company may not need all the topics in the model policy at the outset; the policy is drafted to allow you to remove the “extras” easily. The most important elements of the policy are the compliance elements, which make up most of the policy. Topics like contributions, code releases, and transactional terms can be added when the company is ready for them; before that time, these elements are usually handled ad hoc.

---

1 Willful infringement can lead to a higher statutory damages award, but that is a relatively minor enhancement compared to what occurs in other areas of law.
Business Processes

The corollary to the policy is the process that attends it. The policy only describes certain checks and processes; the business then has to implement them. For example, if using GPL code in a product requires the approval of an engineering manager or the legal department, the company may want to implement an automated way to generate that request. Also, every company needs to consider how it will capture and store the information about the open source software it is using. Using spreadsheets is a fairly bad way to do this, and any method is only useful if it is easy to update and if updating it is incorporated into the development and release cycle. A company should also consider when in the development cycle the open source “seal of approval” needs to be applied. The closer to release it happens, the more likely that any remediation or substitution of code to comply with open source licenses will delay release or simply be abandoned in favor of meeting the release date.

Staffing

The title of the person designated to approve requests to use open source software in the business will differ from company to company. Some companies want to conserve the resources of their legal department and delegate some decisions to engineering management; some take the opposite approach and delegate the question to legal; some have a committee to deal with open source matters (usually composed of representatives from legal, engineering, and management areas); some hire a specialist paralegal or lawyer to handle open source matters. Most open source compliance matters can be handled by someone with minimal, but solid, engineering and legal training. As the volume of approvals at a company goes up, decisions need to be pushed down to an operational level so senior management is not tasked with making routine or recurring decisions. Recurring requests to use the same open source software are frequent because code that works well in one company product may very well work well in all of them.

License-Based Review

The model policy focuses on review of software based on the license that covers it, and this is the way most companies review their potential use of open source software. However, there are good reasons to take a more nuanced approach. Some companies review and approve the use of open source software package by package; this is most common when the company has heightened security concerns or exacting product qualification requirements from their customers.
Some companies create a “sandbox” from which all open source software used in the company must be drawn, and they forbid downloading directly from the Internet. Some companies review open source software for patent infringement or other legal matters (such as export restrictions) as well. Accordingly, the model policy takes a lightweight approach to review, based only on the license.

**Use Cases**

A policy for an SAAS company may not fit a consumer electronics company, and vice versa. They are at opposite ends of the spectrum of open source risk. Depending on the company's appetite for risk, it may require much less review of open source risks than do other companies. However, SAAS companies should keep in mind that basing an open source compliance policy on the absence of distribution can lead to terrible results. Most SAAS companies eventually distribute their products—whether to customers who want a private instance of the software to manage regulatory or security issues, to corporate partners or affiliates, or to successors in interest. So don't start what you can't finish—interleaving GPL code with proprietary code in the same program, particularly at a granular level, will likely lead to problems someday. Also, companies that assume they are immune to open source risk will be tempted to ignore record keeping. Taken together, these practices can mean the software can never be distributed. Then, when the company decides that distribution is its next business initiative, no one will want to deliver the news that it can't be done without a massive reengineering and audit effort.
While most audits are conducted at the time of an investment, corporate sale, or other such transaction, other audits are conducted in connection with customer sales or simply as a process of establishing responsible internal business controls that manage intellectual property hygiene. Today, many customers who are buying licenses for software insist on a full disclosure of open source components. In fact, if the product is to be distributed to the customer, the vendor has an obligation to provide this information to the customer—in the form of the license notices required by the applicable open source licenses.

Open source compliance can be easy to forget and can suddenly become a major concern. It is best to be ready for this challenge before you need to make guarantees that your company is compliant.

**The Challenges of Open Source Compliance**

Open source compliance is not especially difficult—particularly compared to compliance requirements for proprietary licenses, which might require constant tracking of the software’s users, servers, or use cases. In fact, on a conceptual level, open source compliance is easy in most cases. Only a small number of use cases raise the potentially complex legal issues discussed in Chapter 6: What Is Distribution? and Chapter 8: The GPL 2 Border Dispute. But open source compliance does involve some information management challenges, and when people say open source compliance is difficult, they are usually referring to this issue.

The information management challenge arises because open source software tends to circumvent normal business processes. In contrast, if you must license software from a proprietary vendor, then you must first pay a license fee. Most businesses have internal controls that capture the payment of fees; if you must write a check to a licensor, you must sign a license agreement, and the license agreement must undergo legal review, which causes a lawyer or contract negotiator to consider whether the uses contemplated by the company will be compliant and
whether the fees to be paid are reasonable in light of the licensing terms. However, open source is always free of charge and can usually be downloaded freely from the Internet without signing any agreement or paying any fee. This means that for most organizations, there are no internal controls that capture the initial use of the software and, therefore, there is no advance consideration of whether the company can comply with the licenses. Open source compliance is left as a “cleanup” matter for a later time.

As a consequence, many companies wake up one day to discover that they do not know what open source software they are using or whether they are compliant with open source licenses. When this happens, companies may initiate an audit. Alternatively, companies are often forced to initiate an audit when a potential buyer or customer demands it.

**Snapshots, Surveys, and Title Searches**

There are two kinds of audits—the snapshot and the ongoing business review. If you have been asked to undertake an audit to support a transaction like an investment or acquisition, you are probably taking a snapshot. The snapshot identifies the open source software in your code base at a particular moment in time; for an acquisition, that would be the moment of closing the transaction. However, companies that are more sophisticated about open source compliance want to make sure that they are always compliant—not just when someone else demands it. Therefore, they conduct diligence on an ongoing basis as code is added or changed in the code base of their products.

An open source audit is like any other audit, in the sense that one is looking for problems but one can never be sure that all problems have been found. The art of auditing involves engaging in efforts that are financially and technically practical to eliminate the most significant problems. The proper level of effort depends on how risky you think a mistake will be, and that in turn depends on the use case for the software you are auditing and your assessment of the existing internal controls used to manage open source licensing compliance.

There is no one right way to conduct an audit. Lawyers are often asked to make a decision about whether to hire a code-scanning consultant to examine the code. It is never wrong to do this. However, hiring a consultant is like buying insurance: it may be useful, but it may be too expensive. It’s possible to spend all the money you like on managing risk, but it only makes sense to spend money on activities that actually reduce risk.

Economically speaking, the equation is this:

\[ AC < R \times L, \]
where AC = Audit Cost, R = Risk of a problem, and L = Loss that would occur if the problem occurred.

Thus, a problem with a small likely loss is not worth spending a great deal of money to find. Risk and loss depend on use case and your assessment of the company’s internal controls. If a company has no internal controls on open source software use, R will be higher. If the software is distributed in a consumer electronics product, both R and L will be higher than they would be for an SAAS product.

It is not difficult to tell whether a company has internal controls for open source compliance processes in place—all you have to do is ask. For instance, if you ask for a list of open source code that is in a product, and if the company doesn’t have such a list (or doesn’t understand the question), then the company likely has no compliance process in place. In such a case, hiring a code scanner is a great idea. However, if the company provides a list of open source code and licenses that looks reasonably professional, you may decide that hiring a code scanner is unlikely to yield additional information that is material to risk.

There are two overarching approaches to automated auditing: forensic matching and string searching. Companies like Black Duck Software, Palamida, nexB, and OpenLogic perform services known as forensic matching. In other words, they examine the source code for a particular code base and match it against known open source code. They identify the known open source code by its tools. The other approach is to run a string-matching program, which programmers often call a GREP. This kind of program compares strings (which in programming language means text characters) in the source code against regular expressions—meaning target strings, sometimes with wild cards—to find indicia of licensing terms such as copyright notices and licensing statements. Using GREPS is a bit like typing a query into a search engine. So for instance, if you searched the code base for “copyright,” “Copr.,” or “license,” you would find copyright notices, licensing notices, and probably a lot of extraneous items as well. Programs like FOSSology, an open source utility for compliance, use this basic approach.

Alternatively, you may be asked to conduct an audit based on the external records kept by the company. Most often, self-disclosures are in the form of spreadsheets or text lists, and they list software and licenses. The difference between these two approaches is like the difference between a land survey and a title search. Forensic matching is like a survey—it looks at what is actually there and tries to map it onto known information. String matching or self-reporting is like a title search—it looks at what people have recorded about the licensing of the code and assumes that the information is reliable.

---

1 GREP, the name of an old UNIX routine that does string searching, stands for “globally search a regular expression and print.”
In truth, neither of these approaches is foolproof, nor is either without its challenges. Forensic matching is less prone to false negatives—it can find open source for which the copyright and licensing notices have been removed from the software being audited. This removal of licensing notices is considered poor programming practice, but it happens. However, forensic matching is prone to false positives in that it finds potential problems that are not actual problems. On the other hand, a GREP approach will not identify code for which licensing information has been removed or not properly preserved. Therefore, the GREP approach is very useful at identifying entire packages, files, or libraries of open source code, but it may not be successful in identifying fragments or snippets of code.

In an auditing effort, false negatives are more dangerous than false positives. Therefore, proper auditing techniques should tend toward zero false negatives and minimal false positives. To understand better how this works, consider the following seeming paradox: Most people who test positive for a serious disease probably don’t have it. Does this mean the test is wrong? No, it means the test is properly designed to avoid false negatives. If the test is positive, more accurate tests will be done, and these tests are intended to weed out the false positives. The first level of testing is performed on a much larger population than subsequent tests, so it produces many more false positives than the true positives found in later testing. Any kind of audit of testing works in this way.

But as a result, reports that are generated by auditing efforts can be maddeningly difficult to review. They tend to include a lot of false positives, or “noise.” Every code base will have small fragments of code that may be taken from elsewhere, but not all such fragments constitute infringement. For very common code lines, it may be difficult to tell the original source. Also, the copyright law will not protect what it calls de minimis use of copyrighted material.

The upshot of all this is that audit reports give you information, not answers.

### Setting the Rules

Once you perform such an audit, you must determine whether the information you have found meets the criteria that you have set for compliance. For many companies, these criteria are embodied in an open source software policy. (For more on open source policies, see Chapter 11: Open Source Policies.) Different companies have different levels of tolerance for risk, and different use cases have different risk profiles. However, one thing is certain: you cannot tell whether an audit has passed your criteria if you don’t have criteria to apply.

Today, most companies conducting audits consider permissive licenses to be of little concern. Of course, distributing code that includes open source software licensed under permissive licenses requires the delivery of a notice, so some basic
compliance is always necessary. However, delivering notices is almost always possible, even if administratively burdensome. Mostly, in audits a company is looking for copyleft software—particularly copyleft software under licenses like GPL and LGPL—because it is concerned that the audited software may have been engineered in a way that causes compliance to be impossible, even if all of the information is known and the license notices are applied. For example, if a proprietary application uses a GPL library, it may be impossible to comply simultaneously with both licenses.

The Perils of Self-Disclosures

Self-disclosures are often skeletal, inaccurate, and misleading. They generally contain correct information about the open source code that is included—at least at a package level. A list of open source components can usually be generated by the build instructions for the software. This package-level information is reasonably reliable because the compiler must be given this information in order to build the product. But this information does not include any license information. In a typical open source self-disclosure, license information is prepared manually. However, it is often very inaccurate due to any or all of the following issues:

- Lack of license information
- No versions for licenses
- Conflicting license information
- Lists software that is not open source

Here is a hypothetical self-disclosure:

Linux...GPL
Jboss...LGPL
GCC...GPL
XYZ...GPL/MIT
Java ...GPL
Adobe...public domain

The problems with this self-disclosure are that there are no license version numbers, one of the items is clearly inaccurate (Adobe is not a specific piece of software and is likely neither open source nor public domain), one item references a dual license without explanation (which usually, but not always, means the licensor has a choice), one item (GCC) is a development tool that is unlikely to be in a product, and Java is generally under GPL+ exception. Correcting self-disclosures
is a long-standing pastime in auditing. Those who do audits for a living see all manner of weird self-disclosure. One of my personal favorites was “GNU BSD.” Therefore, if you are conducting an audit based on a self-disclosure, you should not rely on the licensing information you have been given, unless you trust that the source of the information knows how to locate and vet the information.

This means that those conducting audits spend a lot of time researching licensing terms, starting from a list of packages. Researching licensing terms can be tricky. For instance, if you have been told that code base includes a piece of software called FOOBAR, your first step would be to search for FOOBAR in a search engine. If you're lucky, you will find a FOOBAR project page that indicates the license terms. At this stage, several things can go wrong: there may be more than one project called FOOBAR, there may be no project called FOOBAR, or the licensing information for FOOBAR may not be clearly indicated on the project web page. Moreover, if you take the trouble to download the FOOBAR source code, you may find that the licensing notices in the project are not the ones that are indicated on the project web page.

Therefore, researching licensing terms is more of an art than a science. Projects that are competently run will clearly indicate licensing terms on the web page, and the licensing terms will match the license notices that are actually in the code when you download it. Projects that are not competently run may have no license terms or inconsistent license terms. The cloud development platform GITHUB is especially known for projects with no licensing terms. Unfortunately, if no licensing terms are applied, the default is that no rights are granted to use the software. In such cases, there is no way to be compliant.

If you are presented with conflicting licensing terms, such as one license identified on the web page for a project and another in the source code COPYING file, you should make a reasoned judgment as to which set of terms is right. If a project seems not to have taken proper care with upstream rights, you may need to try to determine where the code came from to determine which licenses must apply. Generally, the license that is actually in the source code will be the more reliable indicator. But there is no magic to this inquiry. Assuming the licensor has made his own due diligence mistakes, you can rely on any license that the author has applied to the software. Placing a license notice on the software is usually enough to indicate that the author has intended to apply those licensing terms.
**Versioning**

Occasionally, the license terms that are provided to you in an audit process may be inaccurate because the project has changed its license terms. In other words, the license in the software notices for the code you are auditing may be correct, but it may not match the license terms in the currently available version of the software. One way to avoid this problem is to ask for version numbers in the disclosure to be audited. License changes almost always coincide with the release of a major update, so it is usually possible to check the licensing from publicly available sources if you have the software version number. It is most common for a project to change to more permissive licensing terms rather than less permissive licensing terms, so this phenomenon does not usually cause audit problems. In fact, if the license that applies to the version of the software in the audited code base is too restrictive, it is possible that moving to a later release will solve the problem.

**Solving Problems**

If you identify a problem in an audit, you should always think about a solution. (This comment is mostly for the benefit of lawyers. Engineers always think about solutions; lawyers sometimes seem to think their job is to find problems and stop there.) Almost every audit problem has either an engineering solution or a licensing solution, or both. Always consider all of these possibilities:

- **Remove.** In any code base, quite a bit of code goes unused. There are various explanations, but the most common is that the code was used in testing but not in the final product. The actual binary of a software product need not contain all the code in the source code repository from which the product is built. Therefore, if you find a licensing problem, your first questions should be “Is this code really in the product? If so, can we remove it?”

- **Replace.** Open source packages are often created as alternatives to popular proprietary software, or vice versa. Therefore, if the open source terms are a problem, you may be able to find a proprietary alternative that you can substitute for the open source package with minimal engineering effort. Often, proprietary substitutes are not expensive—after all, there is a substitute good that is free of charge.

- **Reengineer.** This is the option everyone wants to avoid because it is expensive and time-consuming. Of course, it is rarely necessary unless the issue involves GPL or LGPL, because these are the licenses whose conditions rely on the means of integration of software.

- **Relicense.** If the open source software is provided under license terms that you cannot comply with for the use case you are auditing, you may ask the
author if he or she is willing to grant an alternative license. Clearly, this approach will not work with most major projects, such as the Linux kernel or Apache web server, which have set licensing terms based on the project’s philosophy. Relicensing works best for individual authors. Keep in mind that if a project has more than one author and you seek an alternative license, you may have to locate and make an agreement with all the authors.

**Audits and Mergers & Acquisitions**

Anyone who has ever been involved in a mergers and acquisitions (M&A) transaction knows that open source audits can be frustrating to accomplish within the confines of such a transaction. Despite the fact that it is more popular than ever to conduct code scans for M&A deals, most due diligence in M&A is still based on self-disclosure. Unfortunately, these disclosures tend to come late in the deal process and be extremely inaccurate, so they tend to create issues too late to do much about them. If you are running a startup business, you can effectively streamline investment transactions and corporate sales by engaging in at least basic open source compliance and keeping records of the open source software you are using. Preauditing your own products can save you a lot of time and trouble and make your company look more professional in the due diligence process. (For more on this, see Chapter 17: Mergers & Acquisitions and Other Transactions.)
Basic open source software licensing

Information to get started understanding open source licensing.

What is source code?

Source code is the text file that a computer programmer uses to write a program. Source code files are processed through a program called a “compiler,” which creates object code or binaries — the machine language files that actually run on your computer. Most programming languages are written this way, including FORTRAN, C, C++ and Java. Some programs are not executed in object code; they are executed by interpreters or in markup languages where the source code is essentially executed as is. These include HTML, PERL, and most BASIC.

What is Open Source Licensing?

Here is a definition of open source.

“Open Source” is sometimes also called “free software,” but as the FSF says, think free speech, not free beer. Open source licensing means licensing of software with the source code available to licensees, so the licensees can make changes to the software. It doesn’t mean all software should be free of charge. Back in the 1970’s, most software programs were licensed in source code form, because most software was developed custom for the client. In the late 1980’s, when PCs began to be standardized, programs became standardized, too. So developers began developing non-custom, off-the-shelf software. They also began distributing object code only,
and keeping the source code from the user. Open source advocates want to reverse this practice, so all users have access to source code.

Why don’t licensors license source code?

Most software developers only license the object code to their programs for two reasons: (1) they do not want competitors to see how to decode their programs, and (2) they do not want licensees to modify the programs, because they are concerned about technical support problems arising from bugs introduced by “do-it-yourselfers.” On the other hand, open source advocates don’t think it’s fair to have to rely on the licensor to make all fixes and changes.

OK, so why does any developer license source code?

Some licensees have the bargaining power to demand it.

Some licensors are required to use open source licensing because they have used code they got under the GPL.

Some licensors believe free software is in the best interests of everyone.

What is GNU? Linux? The GPL? The FSF?

There is plenty of information on the Web about this. Start with www.fsf.org, the site for the Free Software Foundation. Linux is an open-source alternative to Unix — the operating system used by many large computer systems. Linux is now used as alternative to many other systems, such as Windows and embedded systems. GNU is a recursive acronym for “GNU’s Not Unix,” the name of the project under which Linux is developed and promoted. The Linux kernel is licensed under the GNU General Public License or GPL.

Are there other open source agreements beside the GPL?
Plenty! Check out Mozilla (my personal favorite), BSD, and Apache.

I don’t have to worry about the GPL do I? I’ve heard it can’t be enforced in court.

*Enforceability.* Most unenforceability arguments are based on the fact that the user never accepts the terms of the GPL. In other words, most people have made an argument based on lack of formation. However, “shrink wrap” and similar unsigned software license agreements are considered enforceable today. So this argument is probably not worth relying on.

*Willingness to Enforce.* Most GPL code (such as the Linux kernel) is owned or licensed by the FSF. The FSF has publicly expressed its willingness to enforce the GPL, and has an active informal enforcement program in place.

*What about MySQL? Did that prove the GPL is enforceable?* The only court case involving the GPL so far (MySQL v. NuSphere) did not involve FSF-owned code. Also, it was primarily a trademark dispute, and was decided on grounds other than the enforceability of the GPL. In any case, the NuSphere product at issue (Gemini) was statically linked to GPL code. This is a far more clear-cut case than most software companies face when making GPL compliance decisions, so a decision in that case would have left many open questions.

*Business Consequences.* For the GPL, the business consequences of engendering the ill will of the development community can be worse than any lawsuit.

*What happens if the FSF sues me? Will I have to lay open my source code?* This question can’t be answered briefly. If you want help assessing the consequences of violating the GPL, you should consult with an attorney.

Can you help my with my open source licensing questions?

*Clients.* If you are looking for advice particular to your situation, you may wish to seek my advice as an attorney. I have lots of experience in consulting on open source questions, and I am reasonably conversant in C++ and other languages. If you are an attorney in a more general practice, you may wish to refer clients to me to get expert advice in this area. You may find this
to be an attractive alternative after you read the GPL — a very difficult agreement to analyze and interpret.

There are lots of software lawyers. Why should I call on you for my open source licensing questions?

If your software lawyer does not get this joke: \texttt{#define ever (;;)} s/he may not know how to analyze, say, the complexity of data sharing of inherited classes to determine whether two dynamically linked C functions must both be licensed under the GPL. (Wish I could take credit for the joke but I can't. It was told to me by a teacher in a programming class.) Also, many lawyers use a traditional approach GPL to compliance analysis — which misses the point entirely. They read the agreement and carefully analyze its language. Usually, this is the right way to interpret a legal agreement, but the GPL is not written in traditional legal style, and analyzing the GPL is not just a legal task — it requires knowledge of programming, and knowledge of the industry.

What is open source science/research/biotech?

It's not very precise to apply the term open source in areas other than software. Open source is about access to source code — information that enables full use of the software. Most other contexts don't have an exact analogy. When people use terms like this, they usually either mean community-developed works or open (i.e. patent-free or RAND licensed) standards. There are even people who talk about open source yoga and religious texts! The idea of openness is a good one, but open source is probably best left to describe software.

SHARE THIS:

Like

Be the first to like this.
What are the most difficult questions in open source licensing?

These are the things my clients and I think about a lot.

What is the scope of derivative works under the GPLv2?

I have often called this the $64,000 question of open source licensing. (Some of you are too young for that reference, but it comes from a game show in the 1960's — back then, that much bought you a house, not just a Mercedes Benz E Class.) There are some relatively easy cases to determine GPL scope, and you will find more about those on the FSF's FAQ about GPLv2 on my links page. But there are other cases, such as those involving Linux kernel drivers, dual processors, virtualization layers, that are more complicated. To analyze these questions, one needs some technical understanding about how software is put together, and also an understanding of the legal principles underlying GPL.

What constitutes distribution under GPL and other copyleft licenses?

It's pretty well understood that the copyleft requirements of most licenses only adhere when one actually distributes copies of code. (That isn't true for all the copyleft licenses, just the most common ones.) But distribution can become a complicated question, particularly when copies are provided to developers, testers, corporate affiliates, and outsource providers. This is a situation where the right legal help can be of great benefit in managing the need to comply with copyleft.
Are there implied patent licenses under GPL and other open source licenses?

Although some open source licenses have express patent licenses, none of them have reservations of rights. So, there may be "implied licenses" granted when one contributes to or distributes open source code. This causes great concern for companies that have patent portfolios to protect. The hard part is not usually patents that would read directly on the code — it's of greater concern how far beyond that an implied license would go. The law on implied licenses is very unclear, and overlaps with notions of patent exhaustion and estoppel.

How do I comply with notice requirements for embedded software?

Notice requirements in open source licenses aren't usually complicated, but they can be an enormous headache. The more complex the product, and the less it looks like application software, the more difficult it is to apply notices consistently and rationally and still be compliant. Companies use a lot of different practices, most of which are a compromise of some sort. Coming up with a notice procedure takes knowledge of open source requirements, but also common sense and a practical approach.

Does open source licensing allow or require me to use trademarks?

Open source licenses are not trademark licenses. But there is a tension between trademark law and open source licensing. Trademark law covers when you can, or must, use the trademark, logo or brand name of a company that is a source for software. It also covers when you must not. Most companies that release open source software should carefully consider what brands or identifiers they will apply to that software, and enunciating transparent policies for trademark use.
What should my company know about open source?

A “cheat sheet” for entrepreneurs, lawyers, and engineers.

Do we need an open source policy?

You have a policy, whether it is written down or not. It could range from “no open source at all” to “anything goes.” The question is: does anyone follow it? Is it sensible for your business? Written policies are useful to communicate your expectations about use of open source in your organization, particularly to outsourced developers or engineers in subsidiaries and affiliates, whom you may not see every day.

My company doesn’t use open source. Why should I care about it?

You are wrong. Your company uses open source. You just don’t know it yet. See question #1 about your policy.

Should my engineers contribute to open source projects on their own time?

It’s likely they are already doing so, but they may need help understanding whether they are encumbering your intellectual property rights or complying with your expectations. Some companies don’t allow their engineers to contribute, but there can be a downside to that. The top
engineers will want to contribute, and by contributing, you may be able to set the direction of open source projects so they will meet your needs.

I heard the GPL isn’t enforceable. Is it OK to violate it?

That’s wrong, and no.

I read the GPL and don’t understand it. Help!

That’s OK, most people don’t. You can’t understand it very well by reading it on its face. (Otherwise there would not be many long FAQs about it on the FSF web site.) Most questions about the meaning of GPL have straightforward answers, but if you are asking, for instance, what is a “deriviative work,” or what is “distribution,” you probably need expert help.

Using open source seems complicated? Is it worth it?

Usually it is, but there is no one answer. Open source licensing works better for some software than other software. The market seems to think it worthwhile — most companies use it.

Open Source Community Over-REACTs to X Rated Code

Recently, Apache re-classified code under Facebook’s “BSD+ Patents” license to “Category X,” effectively banning it from future contributions to Apache Foundation projects. The move has re-ignited controversy over the patent grant, but like many events in the open source community, the controversy is more partisan than practical. In fact, it’s unlikely the move will affect adoption of ReactJS, and the criticisms of the BSD+patent grant mostly don’t survive the scrutiny of reason.

The Facebook patent grant, officially called the **Additional Grant of Patent Rights Version 2**, has been in effect for years. It applies to the wildly popular ReactJS code — a Javascript library for rendering user interfaces. The roster of major technology companies using the code is impressive, including such consumer-facing giants as Netflix — and of course, Facebook itself.

A New Reaction to an Old Grant

The reaction to this news is surprising, given the parallel patent licensing model is nothing new. Facebook released its “BSD+Patents” grant in 2013 (with a revision in 2015). But a similar model was used with some fanfare by Google with its WebM codec in 2010. This licensing model involves two parallel and simultaneous grants of rights: a BSD license to the copyright in the software, and a separate grant to practice patents that read on the software. Putting the two together means there are two independent and parallel grants of rights. In this respect, it is quite similar to the Apache 2.0 license which, like BSD, is a permissive license, and which also contains a defensive termination provision that exists alongside the copyright license grant.
Much of the reaction to Apache Foundation’s announcement has just created confusion, such as this article misleadingly calling it “booby-trapped.” In fact, many open source licenses have defensive termination provisions — which are mostly considered a reasonable mechanism to discourage patent lawsuits, rather than a booby trap. They are also the rule rather than the exception; all major open source licenses with patent grants also have defensive termination provisions — each with slightly different terms. The difference between the Facebook grant, which Apache has rejected, and the Apache 2.0 license, which Apache requires for its projects, is more subtle than the controversy suggests.

Defensive Termination Provisions Come in Many Flavors

Defensive termination provisions vary in two main ways: the trigger for termination, and the scope of rights terminated. As to the scope of rights terminated, there are two camps: those that terminate only the patent rights grant (including Apache 2.0, Eclipse Public License, and the Facebook grant) and those that also terminate the copyright license as well (Mozilla Public License and GPL 3). In other words, for most licenses, bringing a patent infringement suit can only cause termination of one’s patent rights; for the others, bringing a patent lawsuit can result in termination of the copyright license as well — forcing one to stop using the code. Copyright license termination is a much stronger anti-patent mechanism, and more risky for private businesses, resulting in some private companies refusing to use GPL3 or MPL code.

The Facebook grant differs from most other open source licenses in its threshold for triggering termination. In Apache 2.0, for example, the termination of the patent grant is triggered by a patent claim accusing the software provided under the license. The idea is to create a “patent commons” for the software. Most other open source licenses follow roughly this calculus. The Facebook patent license also terminates if the licensee brings a claim against Facebook, or against any party accusing a Facebook product. In that respect, the termination trigger is similar to the one in the Common Public License 1.0, written many years ago by IBM. (“If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software... then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed”)
Nothing New Under the Sun

Defensive termination provisions of the scope in the Facebook grant are very common in patent licensing, outside of the open source landscape. Most patent licenses terminate if the licensee bring patent claims against the licensor. The reason is that a licensor does not want to be unilaterally “disarmed” in a patent battle. Most patents are only used defensively — asserted when a competitor sues the patent owner. A sues B and then B sues A, resulting in mutually assured destruction. If B has released its software under an open source license without a broad defensive termination provision, B is potentially without recourse, and has paid a high price for its open source code release. A gets to simultaneously free ride on B’s software development and sue B for patent infringement.

Finally, the Facebook grant itself is not new. The grant was released in 2013, and ReactJS’ popularity has been growing since then. As with many open source licenses, the industry’s willingness to absorb a new license depends on the tastiness of the code released under it. In the case of ReactJS, the code was great, and the patent license terms were new, but reasonable.

Is it Open Source?

Some have suggested that the BSD+Patents Clause violates the Open Source Definition. The OSD does not allow licenses that discriminate against persons or groups, or fields of endeavor. But the patent grant does not have license scope limitations; it terminates if the licensee misbehaves — that misbehavior having a lower threshold for actions against the code author than for others. So it seems likely that BSD+Patents does not violate the OSD, and moreover, CPL is already approved by the Open Source Initiative as compliant. CPL, like BSD+Patents, sets a lower threshold for termination based on patent suits against the code author.

What is the Upshot?

The practical result of the Apache Foundation’s decision is unclear. Category X licensed code cannot be included in an Apache Foundation repository. (That category also includes licenses like GPL.) Apache’s re-classification doesn’t mean anyone is restricted from using ReactJS —
it just can’t be committed in an Apache project. It’s not even clear that an Apache project cannot contain a dependency on BSD+Patents licensed code.

Meanwhile, in private business, there is little controversy about using code under the BSD+patent terms. Most companies have examined the marginal legal risk of this license compared to others (like Apache 2.0) and considered it underwhelming. Unless a company decides to sue Facebook (or accuse its products), the termination trigger has no actual effect. If you want to fling patent claims at a company that developed and released a great piece of code, removing the code from your business seems like a reasonable price to pay.

Some of the controversy seems to arise from concern that Facebook is advantaged over others in the license terms. But that is not the same as harming the open source community. The BSD+patents grant establishes the same “patent commons” as Apache 2.0, as a baseline, but provides more protection for the contributor (Facebook) against software patent claims of licensees. It’s odd that a community so opposed to software patents would find this objectionable, particularly in light of the array of defensive termination provisions that have been used in the past.

**PLEASE NOTE:** This blog entry is about the BSD+patent license, not about Facebook. This post represents my personal views only, and not the views of Facebook. I do not represent Facebook on open source matters, but I did not draft the BSD+patents license grant.

SHARE THIS:

Like

Be the first to like this.

RELATED:

* [Google, WebM and BSD](#)
* [Test your knowledge of open source licensing](#)
* [Android Patent Peace (PAX)](#)

AUGUST 19, 2017

Patrick McHardy and copyright profiteering

Many developers in the Linux community have concerns about the activities of Patrick McHardy. Here are answers to common questions.

24 Aug 2017  |  Heather Meeker (/users/hmeeker)  |  17  |  5 comments

https://opensource.com/article/17/8/patrick-mchardy-and-copyright-profiteering
Many in the open source community have expressed concern about the activities of Patrick McHardy in enforcing the GNU General Public License (GPL) (https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html) against Linux distributors. Below are answers to common questions, based on public information related to his activities, and some of the legal principles that underlie open source compliance enforcement.

Who is Patrick McHardy? McHardy is the former chair of the Netfilter core development team. Netfilter is a utility in the Linux kernel that performs various network functions, such as facilitating Network Address Translation (NAT)—the process of converting an Internet protocol address into another IP address. Controlling network traffic is important to maintain the security of a Linux system.

How much has McHardy contributed to Linux? This is not an easy question to answer. First, it's not easy to assess the importance of contributions; all we can do is look at number and size of commits. And second, even if one tracks commits, the tracking mechanisms are not perfect. Git has a blame feature that tracks who nominally commits certain
lines of code to the git repository. Tools like cregit (https://cregit.linuxsources.org/) can be used with git blame to report commits at a more granular level of a code token, producing a more accurate picture of contributions at a file level. Git blame and cregit are useful because they both use publicly available information—the information just needs to be interpreted properly.

An analysis of blame with cregit can help assess McHardy's potential contributions. For example:

- The bulk of his contributions appear to be concentrated during the period 2006-08 and 2012.
- Of approximately 135 files in which McHardy included his copyright notice, only 1/3 are files to which McHardy contributed 50% or more of the file's code.
- His contributions appear to constitute well under .25% of the code in the kernel.

Most of McHardy's contributions appear to be to Netfilter; however, blame might not always tell the whole story. For example, a committer can check in many lines of code having made only minor changes, or can check in code written or owned by others. For these reasons, the authorship of a committer can be under- or over-reported.

Records of contributions to the kernel prior to 2002 are not useful to identify contributors, because at that time, Linus Torvalds checked in all code. Patrick McHardy's contributions did not begin until 2004.

The difficulty of establishing copyright ownership using development repository metadata arose in the Hellwig v. VMware case (https://www.theregister.co.uk/2016/06/15/vmware_survives_gpl_breach_case/). Courts may be reluctant to accept such information as evidence of authorship.
What copyright rights does McHardy have in the Linux kernel?

Copyright ownership in large projects such as the Linux kernel is complicated. It’s like a patchwork quilt. When developers contribute to the kernel, they don’t sign any contribution agreement or assignment of copyright. The GPL covers their contributions, and the recipient of a copy of the software gets a license, under GPL, directly from all the authors. (The kernel project uses a document called a Developer Certificate of Origin, which does not grant any copyright license.) The contributors’ individual rights exist side-by-side with rights in the project as a whole. So, an author like McHardy would generally own the copyright in the contributions he created, but not in the whole kernel.

What is "community enforcement"? Because the ownership of large projects like the Linux kernel is often spread out among many authors, individual owners can take enforcement actions that are inconsistent with the objectives of the community. While the community may have a range of views on how best to encourage adherence to the GPL’s terms, most agree that enforcement should be informal (not via lawsuits) and that the primary goal should be compliance (rather than penalties). Software Freedom Conservancy (https://sfconservancy.org/), for example, has issued certain principles of community enforcement (https://sfconservancy.org/copyleft-compliance/principles.html), which prioritize compliance over the pursuit of lawsuits or money damages. There is no bright-line rule for when informal actions should become lawsuits, or how much money an enforcer should request. Most developers in the Linux community, however, consider lawsuits only the last resort, and are willing to refrain from legal action and work with users who sincerely wish to comply.

Why have so many open source lawsuits been filed in Germany?

Some plaintiffs seeking to enforce open source licenses have filed their claims in Germany’s court system. There are a few instruments for pursuing legal action in Germany that don’t have exact analogs in the U.S. or other common law countries.
**Abmahnung** ([https://en.wikipedia.org/wiki/Abmahnung](https://en.wikipedia.org/wiki/Abmahnung)) ("warning"): The "warning" is a request from the claimant to the defendant to stop doing something. In the copyright context, it is a letter from the copyright owner requesting that an alleged infringer stop infringing. These letters are issued by lawyers, not courts, and are often the first step in a copyright enforcement action in Germany. In the U.S., the closest analog would be a cease and desist letter.

**Unterlassungserklärung** ([https://de.wikipedia.org/wiki/Unterlassungserklarung](https://de.wikipedia.org/wiki/Unterlassungserklarung)) ("cease and desist declaration" or "declaration of injunction"): The "warnings" will often have a "cease and desist declaration" attached to them. This "declaration" is like a contract—signing it will subject the defendant to legal obligations that might not otherwise exist. In particular, the declaration may contain obligations that are not required by the GPL itself. In Germany, it is common for such a document to contain penalties for noncompliance. In the U.S., the analog would be a settlement agreement, but settlement agreements rarely specify the penalties for breach—and in fact, in the U.S., "penalties" may not be enforceable in contracts. The "declaration" is not a court order, but if the defendant signs it, it may gain the legal force of a court order. **So, signing them before seeking legal advice is often not a good idea.** There are other approaches to consider in dealing with a complainant who sends a cease and desist declaration, including proposing a revised declaration with lesser penalties or obligations. Further, because a cease and desist declaration may also contain a non-disclosure requirement, signing one of these documents may also create additional difficulties, such as restricting the ability to seek support from other defendants or to alert the community about the claimant's assertions.

For details, see [abmahnung.org/unterlassungserklaerung/](http://www.abmahnung.org/unterlassungserklaerung/).
Einstweilige Verfügung
(“interim injunction” or “preliminary injunction”): The “interim injunction”
is a court order that is like a temporary restraining order in the U.S. A
defendant’s non-response to a “warning” or “declaration” can encourage a
plaintiff to seek an “interim injunction,” although there is no requirement that
a claimant send a “warning” before requesting an “interim injunction” from a
court. Interim injunctions for copyright infringement can prescribe penalties
of 250,000 Euro or 6 months’ imprisonment. In the U.S., in contrast,
criminal penalties for copyright infringement are extremely rare, and must
be pursued by the government, not private parties. Also, in the U.S., courts
do not prescribe remedies for future possible infringement—they only order
defendants to stop current infringement or pay damages. In Germany,
interim injunctions are also available ex parte, meaning that a plaintiff can
apply to the court without the defendant being heard, and they can issue
without the defendant’s participation. If you receive a “warning,” and
suspect that a request for an “interim injunction” might follow, there is a
possibility to file a preemptive “opposition” with the court.

For details, see Abmahnung.org (http://www.abmahnung.org/einstweilige-
verfuegung/).

Widerspruch (https://de.wikipedia.org/wiki/Widerspruch_%28Recht%
29#Zivilprozessrecht) (“opposition” or “contradiction”): The “opposition”
is an opportunity for a defendant to file an opinion with the court that an
“interim injunction” is not justified.

For an example of a case in which this process took place, see this English
translation of a German court order

How many claims has McHardy brought? Due to the lack of publicly
accessible records for many German court r concoct 3, it is difficult to
determine the precise number of actions brought by McHardy. It has been stated that McHardy has approached over 50 enforcement targets. For details, see Source Code Control (https://sourcecodecontrol.co/gpl/) and 7 Notable Legal Developments in Open Source in 2016 (https://opensource.com/article/17/1/yearbook-7-notable-legal-developments-2016). That doesn’t necessarily mean 50 lawsuits—it probably means 50 demands threatening a lawsuit. But it is difficult to verify this claim with public sources. For details, see Litigation and Compliance in the Open Source Ecosystem (https://www.slideshare.net/blackducksoftware/litigation-and-compliance-in-the-open-source-ecosystem).

Why hasn’t the community stopped McHardy? Various members of the community, including Software Freedom Conservancy, have reached out to try to persuade McHardy to change his strategy, but thus far they have not been successful. The Netfilter project recently published a licensing FAQ (http://www.netfilter.org/licensing.html) addressing concerns about McHardy’s actions.

What can we do to stop McHardy and other copyright profiteers? There is no one answer to this question, and there may be no way to completely stop them. But here are some suggestions for what might reduce the number of copyright profiteers.

Strive to comply with open source licenses. There are plenty of resources to learn how to comply with licenses, and how to set up an open source compliance program at your company. For example:


• The OpenChain project (https://www.openchainproject.org/) publishes a specification for recommended internal processes for open source management.

Don’t sign an Unterlassungserklärung (https://de.wikipedia.org/wiki/Unterlassungserklärung) before seeking legal advice. As explained above, an Unterlassungserklärung can subject you to obligations and penalties that are not found in the GPL itself. Don’t cooperate with the profiteers. You can make yourself a harder target, and enlist the help of other targets in the community.

Support open source development. Authors should not have to resort to profiteering to make a living. Companies that use open source software should not expect open source developers to develop software for free; they should chip in to support important projects.

Learn to recognize a copyright profiteer. Be aware of the general differences between community-oriented GPL enforcement and copyright profiteering. Community-oriented enforcement generally aims to achieve GPL compliance through education and assistance, while respecting users’ freedoms. Profiteering, by contrast, may focus on poorly researched scattershot claims and the threat of legal action for purposes of financial gain. Be on the lookout for assertions that prioritize financial gain and set the stage for unreasonable damages penalties.

Make claims public. If you are the target of a profiteer, and have a choice to make the claims public, doing so might help both you and others by discouraging their actions. As members of the open source community, we all share a duty to speak out against profiteers who seek to burden the
community with allegations that can be resolved in more appropriate and less contentious ways.

Topics: Law (/tags/law)

---

About the author

**Heather Meeker** - Heather Meeker is a partner in O'Melveny & Myers' Silicon Valley office. She advises clients on technology transactions and intellectual property matters. She is an internationally-known specialist in open source software licensing. She received the prestigious IP Vanguard Award for private practice from the Intellectual Property Section of the California state bar for 2016. Best Lawyers named her the IT lawyer of the year for 2018.

* More about me (/users/hmeeker)
* Learn how you can contribute (/participate)

---

Recommended reading

- Don't over-React to the Facebook patents license
  (/article/17/9/facebook-patents-license?utm_campaign=intrel)
- An economically efficient model for open source software license compliance
  (/article/17/9/economically-freedom?utm_campaign=intrel)
- We don't make software for free, we make it for freedom
  (/article/17/8/software-freedom?utm_campaign=intrel)

Court Upholds Enforceability of Open Source Licenses

May 3, 2017

The District Court for the Northern District of California recently issued an opinion that is being hailed as a victory for open source software. In this case, the court denied a motion to dismiss a lawsuit alleging violation of an open source software license, paving the way for further action enforcing the conditions of the GNU General Public License ("GPL").

Artifex and the Dual Licensing Model
Plaintiff Artifex Software, Inc. provides Ghostscript—an interpreter for the PostScript language and the Adobe Portable Document Format (PDF). This popular software is available under a choice of licensing options, under a model often referred to as dual licensing. This business model was pioneered by MySQL AB in the 1990s and later adopted by many businesses seeking to maintain a balance between developing free software and funding development with license fees.

Under its dual licensing model, Artifex offers Ghostscript under two sets of license terms. The GPL—most famous for applying to the Linux kernel—is an open source software license that allows free modification and redistribution of software, subject to making the source code available under GPL license terms, among other requirements. This "copyleft" rubric ensures that all recipients of the software have free access to the source code. Artifex also grants alternative licenses on proprietary licensing terms, for companies that prefer not to comply with the requirements of GPL.

Hancom, Inc. integrated Ghostscript into its software product. It did not seek a proprietary license, but it also did not comply with the conditions of GPL. Artifex then tried to convince Hancom to comply with the GPL conditions, and when Hancom refused, Artifex filed suit to enforce its rights and vindicate the GPL regime.

**The District Court Affirms That Artifex’s Case Can Proceed**

Artifex’s complaint asserts that Hancom’s violations of the conditions of the GPL constitute both copyright infringement and breach of contract. Artifex has requested remedies including compensatory, consequential, and statutory damages, as well as attorneys’ fees and costs. Artifex also sought injunctive relief barring further infringement by Hancom and requiring Hancom to comply with obligations under the GPL.
Hancom moved to dismiss Artifex’s complaint on several grounds. The District Court denied Hancom’s motion to dismiss on each ground. A few aspects of the decision are of particular interest to the open source community. For example, Hancom argued that Artifex could not plead breach of contract for violation of GPL and could not request specific performance of the terms of GPL. Hancom also argued that copyright damages were not available because the GPL grants royalty-free rights.

**Open Source Licenses as Contracts and Damages for GPL Violations**

As part of its motion to dismiss, Hancom argued that using open source code offered under open source licensing terms does not form a contract. Whether open source licenses can be contracts in addition to conditional licenses has been an unsettled area of law. In the seminal case on enforcement of open source licenses in the United States, *Jacobsen v. Katzer*, the Federal Circuit Court of Appeals held that open source violations could be brought as copyright claims, but did not foreclose the possibility of bringing contract claims as well. In *Artifex*, the District Court ruled that Artifex’s breach of contract claim could proceed, finding that the GPL, by its express terms, requires that third parties agree to the GPL’s obligations if they distribute the open-source-licensed software.

Hancom also argued that because the grant of license under the GPL is royalty free, Artifex could not plead damages for a violation of the GPL. Relying on *Jacobsen v. Katzer*, the District Court concluded that royalty-free licensing under open source conditions does not preclude a claim for damages.

**Implications for Open Source Software Licensing**

Here, in denying a motion to dismiss, the District Court only holds that the claims may proceed on the theories enunciated by Artifex, not necessarily that they will ultimately succeed. Still, the case represents a significant step forward for open source plaintiffs. Many open source compliance claims have
been brought as copyright infringement claims, and Jacobsen affirmed this approach. Generally, copyright claims may afford plaintiffs more damages and stronger remedies than contract claims. However, contract claims may help a plaintiff pursue a violator’s worldwide conduct in a way that jurisdictional limits on copyright claims might not allow. Breach of contract claims may also be able to address reputational harm and other indirect non-economic benefits that a plaintiff might derive from enforcing open source license conditions. A breach of contract claim might also, in certain instances, allow for specific performance of open source obligations.

In the past decade, while enforcement of open source licensing violations has become more common, few enforcement cases result in published law. The open source community will be watching this case carefully, and this initial decision vindicates the rights of the open source authors to enforce GPL terms on both breach of contract and copyright theories.

---


2 535 F.3d 1373 (Fed. Cir. 2008).

This memorandum is a summary for general information and discussion only and may be considered an advertisement for certain purposes. It is not a full analysis of the matters presented, may not be relied upon as legal advice, and does not purport to represent the views of our clients or the Firm. Darin Snyder, an O'Melveny partner licensed to practice law in California, Melody Drummond Hansen, an O'Melveny partner licensed to practice law in California, Washington, D.C., and Illinois, Heather Meek, an O'Melveny partner licensed to practice law in California, and Jesse Koehler, an O'Melveny associate licensed to practice law in California, contributed to the
content of this newsletter. The views expressed in this newsletter are the views of the authors except as otherwise noted.

Portions of this communication may contain attorney advertising. Prior results do not guarantee a similar outcome. Please direct all inquiries regarding New York's Rules of Professional Conduct to O'Melveny & Myers LLP, Times Square Tower, 7 Times Square, New York, NY, 10036, Phone:+1-212-326-2000. © 2016 O'Melveny & Myers LLP. All Rights Reserved.

Key Contacts

Darin Snyder
San Francisco
D: +1-415-984-8846

Melody Drummond Hansen
Silicon Valley

D: +1-650-473-2636

Heather J. Meeker
Silicon Valley
D: +1-650-473-2635

Jesse J. Koehler
San Francisco
D: +1-415-984-8715
Related Practices

Intellectual Property & Technology
“Stand Up for your Product”: Allocating Liability for Open Source Software
Who is has the “moral high ground”? 

Vendor – See Indemnity as Tax

Customer – Sees Indemnity as Protection
But indemnities are not about what is right, but what is commercially reasonable.

Economics – What is the most efficient way to avoid or resolve problems?
Non Compliance
How is Open Source different?

Embedded Software

Docker images and containers

Base image file

Dockerfile

FROM xxx
ADD xyz
RUN foo
CMD bar

Image file

Docker "build"

Running container

Docker "run"
Modified UCC Approach

• § 2-312. Warranty of Title and Against Infringement; Buyer's Obligation Against Infringement.
What is a Product?

Age-Appropriate Explanations
What is a product? (Kindergarten)
What is a product? (Third Grade)
What is a product? (Seventh Grade)
What is a product? (High School)
What is a product? (Real World – Example)
When you Stand Up for your Product

What are you standing up for?

Certainly this

Probably this

Not this

Application

Library

Language Engine

CMS

Operating System
How do we separate the stack and the product?

Technical approach...

- If the open source is named in the specifications for the product, it is not covered by the IP indemnity
- If the specifications for the product require use of the open source, there is no indemnity
- If the specifications for the product don’t require it, and the vendor selects it as part of an internal build-or-buy decisions, vendor indemnifies
- When a company makes goods to a buyer’s specifications, the buyer — not the seller — provides the warranty of non-infringement (U.C.C. 2-312(3)).
How do we separate the stack and the product?

Various Approaches

- **Interoperability**: The product excludes any code communicating with the application via standard or public APIs (i.e. that can be swapped out with other code).

- **GPL Approach**: The product includes everything in its executable (i.e. that cannot run on its own).

- **Source of Code**: The product excludes anything not written by the vendor.

- **Linux Definition**: The product excludes everything in the Linux stack as defined by the Open Invention Network.
What does the UCC say about IP Indemnities?

• § 2-312. Warranty of Title and Against Infringement; Buyer's Obligation Against Infringement.

• (3) Unless otherwise agreed a seller who is a merchant regularly dealing in goods of the kind warrants that the goods shall be delivered free of the rightful claim of any third person by way of infringement or the like but a buyer who furnishes specifications to the seller must hold the seller harmless against any such claim which arises out of compliance with the specifications.
How do we separate the stack and the product?

### Modified UCC Approach

| “Open Source Computing Stack” means any open source software created by third parties that is referenced in the specifications for the computing environment of the Product [in Exhibit ____], which software may include operating systems such as Linux, web server software such as the Apache web server, language engines such as Java, PHP, Python or PERL, and database software such as MySQL.

- Vendor will have no liability under [reference indemnity provision] for infringement of third party rights by the use of the Open Source Computing Stack; provided, however, that the foregoing sentence will not limit Vendor’s liability for compliance by Vendor with the terms and conditions of the open source licenses applicable to the Open Source Computing Stack.
Who solves an infringement problem with Linux?
Thank You for Your Time!

Heather Meeker
hmeeeker@omm.com
+1-510-463-1116

My blog (for news on open source licensing and other goodies)
heathermeeker.com

Second Edition now available!