1 of 1 DOCUMENT

The Computer & Internet Lawyer

March, 2003

SECTION: SOFTWARE LICENSING; Vol. 20, No. 3; Pg. 20

LENGTH: 5411 words

HEADLINE: **Open Source, Free Software,** and the General Public License

BYLINE: by Jason B. Wacha; Jason B. Wacha is Vice President of Corporate Affairs and General Counsel of Monta-Vista Software in Sunnyvale, CA.

BODY:

People often hear the term "free software" and think that they do not have to pay for it. Some believe that applications running on the Linux operating system cannot be proprietary and that such applications are subject to having their source code disclosed to world. These are common misconceptions about "open source" software.

This article will address the background and present legal status of open source software licensing and the General Public License (GPL). In addition, it will introduce some of the basic engineering concepts that often are an integral part of the analysis of whether code must be licensed under the GPL. Any article on open source and the GPL, however, is likely to be too short for its subject, a target for widely diverging opinions, and quite possibly outdated in part by the time it is published. Such is the nature of one of the fastest growing, ever adapting, and most commonly misunderstood licensing schemes in the world. As for law on this topic, in many instances, we are still trying to figure it all out.

Keeping these principles in mind, this article is intended to leave the reader with a basic understanding of open source licensing as it stands today from two standpoints: technical and legal. A detailed exploration of these topics, and of the technology issues in particular, could be the subject of a long treatise and is daily the subject of wide-ranging discussions on the Web and throughout the open source community. But if the reader of this article remembers just two points -- that "free software" does not mean *free* in the monetary sense and that proprietary code can and does happily co-exist with GPL code -- then this article will be a success.

Why Is Open Source Important?

Your company, or your client, is probably running much open (and free) software right now. So what kinds of programs are "open source"? One of the most widely used Web server programs in the world (Apache), one of the dominant Web programming languages (Perl), the program that routes more than 80 percent of all Internet email messages worldwide (Sendmail), the program that is the basis for the domain name system (BIND), and the fastest growing operating system in the world (Linux) are all open source. Many of the biggest high-tech companies in the world have committed significant funds or based new product lines on Linux and other GPL code. A partial list includes Alchemy, AMD, Compaq, Ericsson, IBM, Intel, Lucent, Motorola, Netscape, Nokia, Nortel, and Sony.

Background and Terminology

Even those experienced in the open source community sometimes use several terms interchangeably: open source, free software, and the GPL. While in some cases the differences in meaning are more political than legal, the distinctions are important.

Free Software

One of the most consistently misunderstood phrases in licensing today is "free software." The most important point to remember is simply that "free" does not refer to money. "Free" refers to liberty, to freedom, rather than to a price tag of zero. It means that a person is pretty much free to do what he or she wants with a program. To use the language of the Free Software Foundation (FSF): "Think free speech, not free beer." A free software user may get the software at no cost or may pay money for it. As long as the user has the freedom to access the source code and to run, copy, study, modify, and redistribute the program (without having to pay any additional money), then the software is free"

For more on the FSF's philosophy on and definition of free software, see *www.gnu.org/philosophy/free-sw.html.* It was, in fact, the confusion over, and misuse of, the term "free" software that helped lead to the coining of the term "open source" and the founding of the open source movement.

Open Source

Open source software has been around for a long time, but the open source movement is a more recent phenomenon. The simplest definition of "open source" is that the source code for the program is not secret or proprietary. In other words, the user can pull up and view the human-readable C, C++, scripts, or other types of instructions that the programmer wrote to create the program.

For all the unfamiliarity with open source software, it is not a new concept. In fact, it has been around for more than 30 years. When computers first appeared on the business scene, they were large and expensive to operate, and software often had to be tailored to each user's needs. n1 Users found that it was cheaper, better, and more efficient to share software. Developers typically shared source code for programs to foster quicker, more efficient, and more effective development. It was only after computers (and accompanying software) became mass-produced that software developers moved to a closed source (proprietary) development model. n2

n1 *See* Daniel B. Ravicher, "Facilitating Collaborative Software Development: The Enforceability of Mass-Market Public Software Licenses," *5 Va J. L. & Tech 11 (2000).*

n2 *Id.*

More recently, various persons and groups have attempted to create a more specific definition of open source and also have tried to distinguish themselves from the free software movement. While the two camps may seem similar, they are politically different animals. In part, the open source community grew out of a split between the ideologies of Richard Stallman n3 and the FSF. The two camps are aligned in perhaps the most important area, however -- both focus on a user's ability to access and redistribute source code.

n3 One of the leaders of the free software movement, founder of the GNU project, calls developing free v. proprietary software a "stark moral choice."

One group, the Open Source Initiative (*www.opensource.org*), has developed a widely used definition labeled the "Open Source Definition," or "OSD." The OSD focuses primarily on free redistribution of programs and the requirement to offer or provide source code. In addition, licenses designed to meet the OSD must include specific requirements regarding distribution of, modifications to, and restrictions on the code. An annotated version of the OSD is available at *www.opensource.org./docs/definition.html.* The OSD is not a license in itself, however; it is merely a definition created by a specific group.

The GPL and the LGPL

What Is the GPL and How Does It Work?

The GPL is a license agreement published and copyrighted by the FSF. The GPL (also sometimes called the GNU GPL) is just one type of license agreement that covers open source and free software. Other such licenses include the BSD n4 license, the Apache license, the Perl license, Netscape's Mozilla license, and many others. The focus of excitement, debate, and confusion seems to center on the GPL, however. Yet, the GPL is a fairly short license, and it lacks many of the complicated provisions that often accompany proprietary licenses.

n4 BSD is an abbreviation for Berkeley Systems Distributions.

Licensing software under the GPL does not mean that the licensor has to give away the software, publish it on the Internet, or charge only minimally for the software. In fact, the GPL covers only three activities: copying, modifying, and distributing a program. All other activities are expressly outside its coverage. n5

n5 GPL § 0.

While the language of the GPL is fairly straightforward, it is all too often misunderstood. Three areas in particular seem to cause the most confusion: (1) whether a party can charge money for GPL code, (2) whether a party has to make source code available for a GPL program, and (3) whether proprietary code can coexist with GPL code. The short answer to all three questions, addressed more fully below, is yes.

What Is the LGPL?

LGPL is the Lesser GPL (formerly the Library GPL). The LGPL, while still promulgated by the FSF, evolved through commercial forces outside the FSF. The LGPL provides a way to use libraries that otherwise might subject a proprietary application to GPL terms. For example, if a developer writes a program and links it to a library licensed under the GPL, the program will be subject to the GPL. If, however, the developer links the program to a library licensed under the LGPL, the program can remain proprietary even though it runs on the Linux operating system. (Linux, a free Unix-type operating system originally created by Linus Torvalds with the assistance of developers around the world, is illustrative because it is the best-known and most widely used operating system licensed under the GPL.) An application is free to include the header files and reference code from an LGPL library as long as the header file obeys usual programming conventions as outlined in the LGPL license. In the context of Linux, only a small number of user program libraries are GPL-licensed, thus greatly increasing the possibilities for developers to write proprietary Linux-based programs. Like the GPL, the LGPL requires that, when a developer delivers object code that falls under the LGPL (for example, by modifying or enhancing an LGPL library module), the developer must either (A) deliver source code or (B) provide a way for the recipient to get the source code.

A Software Provider CAN Charge Money for GPL Programs

GPL programs are "free software." Remember, though, that "free" doesn't refer to money. Sections 1, 2(b), and 3(b) of the GPL provide that, if a software provider has a GPL program on a disk and someone wants to obtain that program, the provider can charge any amount for the service of delivering it. n6 Could the potential user get the program somewhere else, possibly even without paying for it? Maybe. If the potential user wants a particular provider to deliver the program, however, that provider can charge for it.

n6 *Id.* § 1; GPL Preamble. (*See also* the FSF Web site, on which it encourages persons to charge money for the delivery of GPL programs.)

Once the user has the program in his possession and regardless of where he obtained it, the provider cannot charge additional money based on use of the program. n7 In other words, royalties are not permitted.

n7 GPL § 2(b).

The language of Section 2(b) of the GPL can be confusing initially: "You must cause any work . . . to be licensed as a whole at no charge to all third parties under the terms of this License." Just remember that the "licensed . . . at no charge" language refers only to the three "licensed" rights -- to copy, modify, and redistribute the program. To the extent that a provider wants to offer a customer additional perks, such as warranties; indemnifications; support; service; updates and upgrades; and revisions, the provider and customer negotiate for those items and services separately. Such terms are wholly outside the GPL.

Requirements for Delivery of Source Code Under the GPL

Once a user has a software provider's GPL program in binary form, the user has the right to receive the source code. This does not mean that the provider must release the source code to the whole world. It means only that the provider must give its purchasers of the software the source code itself or the right, exercisable for three years, to receive the program source code. Many software providers choose to include the source code with the initial delivery of binaries, but this is not required. If the provider does not deliver the source code initially and a customer subsequently requests it, the provider must charge no more than the cost of delivery. n8 In other words, while a provider can charge any amount for binaries, it cannot provide binaries (free or at any cost) and then surprise the customer with a huge bill for the source code.

n8 *Id.* § 3(b).

The Interaction of Proprietary and GPL Code

The area open to the broadest interpretation, and the most intense debate, surrounds when and how proprietary code can coexist with GPL code. An in-depth discussion of this topic merits an article unto itself. A brief overview here is worthwhile, however, to stress an important point: A developer can write a proprietary program, run it on a GPL database (such as Linux), and retain the proprietary rights in the program (including keeping the source code secret and controlling the redistribution of the program). Middleware and applications running on a Linux distribution (such as MontaVista Linux) can remain proprietary, just as they remain proprietary when running on top of other operating systems (such as Windows, DOS, Unix, or others).

When determining whether a new program can run on a GPL-licensed platform (such as Linux) and still remain proprietary, the developer needs to consider three primary factors: (1) whether the program is an original work, (2) how the program interacts with GPL code, and (3) whether the program operates in the user-space domain (such as word processing programs, productivity software, or whatever else the particular program *does*) or in the kernel-space domain (as in a driver or other module that extends kernel functionality).

Origin of the Code

The first thing a developer must examine is the origin of the code in the program. To keep the program proprietary, the work should be original. If the program is "derived" from work that is already subject to the GPL, then the developer must license the work under the GPL. The exact definition of a "derived work" for GPL purposes has not been established, but two possible sources for guidance exist.

First, refer to the definition of a derived work under the Copyright Act:

a work based upon one or more preexisting works such as a translation . . . abridgment, condensation or any other form in which a work may be recast, transformed or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work or authorship, is a 'derivative work'. n9

n9 *17 U.S.C. § 101.*

Second, and more importantly for purposes of the GPL community, use some common sense. If it appears that a developer is trying to use code that is subject to the GPL and turn it into its own proprietary code, the community likely will see the developer as violating the terms of the GPL. Each of the analyses below assumes that the developer has passed the original code hurdle.

Interaction Between Programs in the User Space

To run effectively, applications must link to other, preexisting code; usually, applications link to a library of software. In determining whether an application can remain proprietary, the licensing status of the library to which the application is linked is of key importance. n10 The accepted practice in the open source community is that an application linked to a GPL library must itself be licensed under the GPL. An application linked to a non-GPL (including LGPL) library can remain proprietary.

n10 With regard to static and dynamic linking, it is becoming accepted in the GPL community that, in the user space, the method of linking is not determinative; it is the status of linked program that generally controls.

Interaction Between Programs in the Linux Kernel Space

Programs written in the Linux kernel space interact with the core of the operating system. A typical program written to operate in the kernel space would be a device driver (to run a printer or interface to embedded control hardware). In the Linux world, determining whether a driver (or other kernel module) can remain proprietary involves an examination of how closely the code interacts with the kernel code. In this regard, there is a distinction between static and dynamic loading. Static loading is typified by a driver that is designed to boot up together with the Linux kernel so that it essentially loads as a single image with the kernel. Conversely, dynamic drivers do not load when the kernel boots up; they load only later, in run time, when they are needed by the user, by a specific application, or by another kernel module.

Linux kernel module analysis is widely debated and opinions are shifting. Currently, however, a general principal is fairly broadly accepted: Statically loaded modules/drivers must be licensed under the GPL, as they directly extend the kernel. Dynamically loaded modules/drivers may retain proprietary status since they are physically and logically separate from the kernel code prior to load.

Why Pay for "Free" Software?

Some may ask whether the same GPL software for which companies charge money is available free on the Web. The answer is yes and no. While the source code can be reconstituted from several hundred sites (or source trees), even a talented software engineer may not succeed in creating an exact image. Given the way Linux exists on the Internet, especially in the embedded Linux world of cross-development and non-X86 n11 hardware, it is extremely complex to build. To create the necessary binaries and host environments, a potential user would want to dedicate a substantial team of engineers working every day in the open source community. Moreover, a user then would need to maintain the code (with more than two million lines in the Linux kernel alone) and update it with new versions of the Linux kernel and patches, among other things.

n11 X86 is a generic name for the series of Intel microprocessor families that began with the 80286 microprocessor. *See www.searchHP.com.*

For most companies, it boils down to a classic make v. buy decision. Even if a potential user could find the same code at no charge somewhere, it is usually more efficient to get that code from expert Linux providers than to develop in-house expertise. Additionally, a potential user who buys code from a vendor often obtains warranties, indemnifications, support, maintenance, upgrades, training, professional services (such as custom software development) by expert developers, and assurances of a reliable, quality-assured binary distribution. To use Eric Raymond's words, the user gets "a continuing exchange of value between vendor and customer." n12

n12 Raymond is best known as the co-founder of the Open Source Initiative (OSI) to promote Linux and other "free" software to businesses. *See* Raymond's writings at *http://www.catb.org/ esr.*

Enforceability of the GPL

The GPL never is signed. It is delivered as a shrinkwrap (and sometimes as a clickwrap) license. The GPL requires that any recipient of a GPL program receive a copy of the GPL "along with the program." n13

n13 GPL Preamble, § 1.

Enforceability as a Shrinkwrap License

At least in the United States, shrinkwrap and clickwrap licenses generally are considered enforceable, at least so long as the user has a chance to view the license before opening the program and takes some physical step to accept the license's terms (such as breaking open a shrinkwrap or clicking "I accept" in a clickwrap). With the GPL, however, neither of those actions usually occurs. So how does a user accept the license's terms?

In theory, the user accepts the terms of the GPL just by doing the things that the license allows. The user does not need to accept the license in order to acquire, install, examine, or use the program. The license applies only if the user wants to copy, modify, or distribute the program. In those instances, the GPL provides that, since the user does not have the right to do any of those things without accepting the license, the very act of exercising any of the three licensed rights indicates that the user has accepted the GPL terms. n14

n14 *Id.* § 5.

Enforceability Generally

The GPL never has been tested in court. Then again, most companies' software license agreements have not been tested in court either. That does not mean, however, that the agreements are not enforceable.

Still, much of the law surrounding the GPL is unsettled. For example, the conventions discussed above in the section entitled "The Interaction of Proprietary and GPL Code" do not come from any settled law or even from the GPL itself. They arise from (1) a community dialogue with Linus Torvalds (and others) and certain commercial vendors who desired to port the drivers to the Linux operating system and (2) accepted practices among Linux developers. That said, what is the "formal" legal status of the GPL?

First, there is the question of standing. Although the FSF authored and copyrighted the GPL, the FSF generally is not a party to it. So a lawsuit for violation of the GPL likely would be between two private parties, just as would a lawsuit over any proprietary license. The FSF can, however, become involved in GPL disputes (and potential disputes) both informally, via telephone calls or meetings with parties, and formally, via *amicus curiae* briefs. Often the informal involvement of the FSF in a potential GPL violation action is enough to resolve the issues and forestall any formal filing. The general counsel of the FSF has stated, "I've been able to enforce [the GPL] dozens of times over nearly ten years, without ever going to court." n15 The FSF, to date, has never insisted on payment of damages. (Note that, if the suit were not simply over a contractual GPL violation but over copyright or other violations with respect to GPL code, then any of the copyright holders whose rights were violated might have standing, and that could include the FSF.) The FSF is also a major source of open source software (like gcc n16 and binutils n17) and, consequently, owns the copyright on key software in the open source community, as well as prime enabling software for Linux itself.

n15 Eben Moglen, "Enforcing the GNU GPL." *See http://www.gnu.org/philosophy/enforcing-gpl.html/.*

n16 GCC is an abbreviation for GNU C Compiler. GNU is a recursive acronym for GNU's not Unix. GNU is the FSF's project to provide a freely distributable replacement for UNIX. *See http://hostingworks.com/support/.*

n17 Binutils is a GNU collection of binary utilities.

One of the more public examples of the FSF's informal involvement in a dispute was with FSMLabs, the author and commercial proprietor of RTLinux, a real-time systems add-on. The FSF worked with FSMLabs to resolve a claim that FSMLabs had violated the GPL by using a patent license to restrict distribution of its modifications to the open-source Linux operating system. Before any formal action was filed, the parties reached an agreement that resulted in FSMLabs releasing a fully GPL-compatible license for the product.

The GPL-related case *Progress Software Corp. v. MySQL AB,* n18 filed in 2001 in federal court in Massachusetts, received significant media attention. Though the primary claims in the case were for trademark infringement, interference with contracts, and unfair competition, the suit also included a claim for breach of the GPL. MySQL claimed that Progress Software's NuSphere Corporation had violated the GPL by distributing programs derived from GPL-licensed programs of MySQL without releasing source code. This case settled out of court in late 2002. n19

n18 Progress Software Corp. v. MySQL AB, No. 01-CV-11031 (D.C. Mass).

n19 "My SQL, NuSphere Settle GPL Contract Dispute," *The Register,* available at *http://www.theregister.co.uk/content/4/28021.html.*

Differences Between GPL-Compatible and Proprietary Licenses

Software licenses compatible with the GPL tend to be shorter and more straightforward than conventional proprietary software licenses. This is due in part to the efforts of the GPL drafters to write a shorter, simpler agreement. Moreover, some provisions of traditional, proprietary licenses are simply inapplicable to the GPL licensing model.

Understanding the Origin of Code

Before looking at warranties or indemnifications, it is important to understand the origin of code. A proprietary vendor typically will get code from one of two places: It will write it, or will license it from a third party. An open source vendor may get code in either of these ways, but in addition, it may gather code from public Web or file transfer protocol (ftp) sites (also known as source trees). If a vendor is also the author of the code, the vendor can state that it has not pirated the code and can provide warranties and indemnities with respect to the code's function and originality. This can be true under both open source and proprietary licensing schemes.

Outside this scenario, the analysis necessarily forks. If a proprietary vendor is passing code from a third-party author, the vendor often is able to pass through any warranties or indemnities that the original author provided. In the open source world, however, most source code is provided "as is" with no warranties and no indemnifications. Thus, there is nothing for the vendor to pass through; any additional protections that the vendor adds are risks wholly taken by that vendor. This becomes even more apparent when a vendor has gathered code from a public source. In this instance, the vendor may not even know the original author of the code and almost certainly will have no protections associated with the code.

Warranties

Individual authors of open source software rarely will provide warranties. The GPL itself expressly states "no warranty" and requires that a vendor pass along to any recipient of the GPL program the disclaimer of warranties. n20 The GPL anticipates, however, that vendors may choose to offer warranty protection for an additional fee. n21 Some commercial Linux vendors have chosen to do so. Some (including this author's employer, MontaVista Software) offer warranties on all the code that they provide, whether written in-house or gathered from outside sources. While it presents some economic risk to the company, it can be an important value added for customers.

n20 GPL § 2.

n21 *Id.*

Indemnifications

The analysis of indemnification provisions is parallel to that of warranties but greater in terms of ultimate economic risk. It is rare for any Linux code provider, including commercial vendors, to offer intellectual property (IP) indemnities. Rare is the vendor (including MontaVista Software) that chooses to add IP indemnities on GPL code as a value add-on for its customers. Such a provision can make the vendor an insurer, however; the decision to offer IP indemnities is as much an economic decision as a legal one.

Escrow Provisions

One of the negotiated provisions in a typical proprietary license is how, when, to what extent, and for how long source code may be released from escrow to the licensee (or if there is going to be an escrow at all). With open source licenses, escrow ceases to be a concern. Normally, source code is delivered to the customer at the time of delivery of

the binaries. If not, the customer still has the right to request and receive the source code at a later time. Because of this immediate and ongoing right of access, the need for any escrow provision is vitiated entirely.

Royalty Provisions

Like escrow provisions, royalties are not an issue in negotiating GPL-compatible licenses. The GPL expressly prohibits charging royalties. n22

n22 *See id.* § 7.

License Grant Provisions and Restrictions

Again, drafting and negotiation time should be shortened considerably for GPL-compatible licenses. Gone are typical proprietary license grant phrases like "limited, non-transferable" and use only for a "permitted purpose" or "at the designated site." Gone are restrictions on copying, maintaining records of copying, selling or sublicensing, or creating derivative works. Gone are prohibitions on decompiling and dissembling and reverse engineering. In short, the GPL greatly simplifies the licensing practices that lawyers and development teams are forced to observe with more restrictive licensing regimes.

Patents and Copyrights Under the GPL

Software programs are both copyrightable and patentable. n23 The authors of the GPL understood this and address both concepts in the license.

n23 Regarding copyright, *see* the 1980 amendments to the Copyright Act.

Copyrights are not a problem in the GPL, and for the most part are obtainable and enforceable just as they are in the proprietary world. The GPL expressly references copyright law n24 and requires authors to "conspicuously and appropriately publish on each copy an appropriate copyright notice." n25 Authors of original code can claim and document their copyright in the program. Persons who subsequently distribute modified or unmodified copies of the program must retain the original author's copyright. The important point to remember is that, in choosing to license under the GPL, a software provider has chosen a licensing scheme that expressly allows others to use, display, redistribute, and modify the program. This is the spirit of "copyleft," the term coined to describe the way in which the GPL model allows persons to use, modify, and redistribute copyrighted source code (or any program derived from that code), as long as the distribution terms remain unchanged.

n24 *See, e.g.,* GPL § 0.

n25 *Id.* § 1.

Patents, because of the requirement to provide a detailed specification that would enable a person of ordinary skill in the art to make and use the program, are tailor made for open source and in fact are difficult (if not impossible) under a closed source model. The GPL authors understood, however, that "any free program is threatened constantly by software patents" that could give proprietary characteristics to a formerly "free" program. n26 Thus, if any author wants to obtain a patent on his or her GPL software, the patent must allow the program to "be licensed for everyone's free use or not licensed [under the GPL] at all." n27 If for some reason, a patent exists that restricts GPL-compliant distribution (if the patent imposes royalties, for example), then the program may not be distributed at all under the GPL's terms.

n26 *Id.* Preamble.

n27 *Id.*

Community, Community, Community

This article has referred to the "communities" of Linux, GPL, open source, and free software. These are not off-hand characterizations or casual references. Because of the still-unsettled "law" surrounding the GPL, the high growth rate of the Linux operating system, and the strong opinions of those persons involved in the worlds of open source and free software, it is the "elaborate set of customs" in the community that provide the strongest guidance (whether politely or otherwise). n28 A software provider that the community perceives is doing "right" may read its praises on the Web. A provider perceived as doing wrong, however, will have negative "flames" spread worldwide. Why else would companies be so quick to comply with informal requests of the FSF? As the FSF general counsel has said: "No one wanted to be seen as the villain who stole free software, and no one wanted to be the customer, business partner or even employee of such a bad actor." n29

n28 *See* Raymond's writings at *http://www.catb.org/ esr.*

n29 *See http://www.gnu.org/philosophy/enforcing-gpl.html.*

Summary

Open source, free software, and the GPL are some of the hottest, and most misunderstood, topics in technology licensing. Yet for all the debate, they boil down to relatively simple concepts. They are neither the death of commercial software nor the end to proprietary programs. They simply describe methods, framed by various legal and community-based conventions, to license software in a manner that can provide economic reward to authors and vendors and at the same time encourage wide distribution, rapid adoption, and freedom to modify and redistribute again. For attorneys, it provides a licensing scheme that, after a little education, can be simpler, quicker, and just as effective as traditional proprietary models. It is also an area that most software licensing attorneys are likely to be exposed to before long. The sooner the education process begins, the better.

LOAD-DATE: August 12, 2003