CHAPTER 2

OPEN SOURCE SOFTWARE: THEORY AND PRACTICE

by John R. Ackermann
NCR Corporation

1611 S. Main Street SDC-3
Dayton, OH 45479
(937) 445-2966
john.ackermann@ncr.com

John R. Ackermann is Law Vice President of the Worldwide Customer Services Law Group at NCR Corporation in Dayton, Ohio. He has practiced in the various disciplines known as "computer law" since 1988 and in his spare time manages a Linux-based network and Internet server in his home.

TABLE OF CONTENTS

CHAPTER 2

OPEN SOURCE SOFTWARE: THEORY AND PRACTICE

John R. Ackermann

I. A BRIEF HISTORY OF OPEN SOURCE SOFTWARE, PART 1

"Open Source" is a philosophy of software development and distribution. Of the two words, "source" is the easy part to explain. It refers to source code, which is the form in which most programs are created. Source code is usually written by and can, at least theoretically, be read and understood by human beings (some source code is actually generated by other computer programs, but we won't go there). Although humans can understand source code, computers can't, and a tool called a compiler is used to convert source code to a binary format (an extremely long string of ones and zeroes) that a computer can digest and execute. The binary version of a program is often called "object code" and an object code version that includes all the pieces necessary to run on a computer is called an "executable."

Source code is important to lawyers as well as programmers because it discloses the software's secrets. No one can look at the millions of ones and zeroes in an object code file and learn much about how the program works. However, a skilled programmer can understand a program by studying its source code and, by modifying the source and recompiling it, can create a modified version of the original. In the traditional software industry, source code is considered almost synonymous with "trade secret," and companies go to great lengths to keep their source code away from prying eyes. Lawyers make their living writing license agreements that protect source code.

However, things weren't always that way. In the days when dinosaurs roamed the machine room (through the 1960s), software normally wasn't considered valuable in and of itself – it was simply something that made having a computer worthwhile. IBM routinely made available the source code for its mainframe computer applications.[1] At the time, it was unclear whether computer programs could even be protected by copyright.

It was the development of an independent software industry in the 1970s that led to the idea of keeping source code a secret. Vendors whose profts came from bits rather than nuts and bolts had no desire to let their competitors, or for that matter their customers, dig into the guts of their products. As the industry grew, it became unheard of to distribute source code.

And that is how things stayed until the mid-1990s. It was taken for granted that distributing source was the surest way to commercial ruin. Where business needs required the disclosure of source code (for example to a hardware vendor porting to a new machine), the required license agreement had non-disclosure terms so stringent that just about all the licensee could do with its employees after they had seen the source code was shoot them.

And that brings us to the "open" in Open Source. Open Source (the term is normally capitalized) is the opposite of this proprietary model of software development. As its name implies, Open Source software is developed through an open process made possible in large part by the Internet. It is shared, and licensed, in source code form, and people are free to modify it as they choose. It is normally, but not invariably, made available to users at no charge. Before talking more about Open Source, which is a development philosophy, it's important to describe the idea of "Free Software" which is a political philosophy. And that political philosophy drives the license that lawyers love to argue about, the General Public License – the "GPL", on which most of this paper will focus.

II. THE PHILOSOPHY OF FREE SOFTWARE

Before continuing with our history of how Open Source became the force that it is, it is necessary to take a slight detour to explain the first of two philosophies that are fundamental to the growth of Open Source software. Although "free software" or "freeware" is often used generically for any software that is distributed without charge, the Free Software Foundation ("FSF"), an organization we'll shortly talk much more about, defines it strictly:

> Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:
>
> - The freedom to run the program, for any purpose (freedom 0).
> - The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
> - The freedom to redistribute copies so you can help your neighbor (freedom 2).
> - The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

(http://www.fsf.org/philosophy/free-sw.html; a page that is well worth your attention if you are interested in the basis of the Free Software movement.)

Free Software by this definition is virtually always Open Source software. But not all Open Source software is Free Software; some Open Source licenses, according to the FSF, do not grant the requisite level of freedom.[2] An acronym that's gained currency in the last year or two is "FOSS" – Free and Open Source Software – to describe the intersection of the two concepts.

The Free Software idea arose from the small group of computer enthusiasts in the academic and hobbyist worlds – let's call them "hackers" in the positive sense of that term – who during the 1980s and 1990s began espousing the libertarian ideas that "information wants to be free" and that source code should be freely distributed. As shown by the FSF's definition, "Free Software" isn't about price, but about freedom.

A commonly used catchphrase is "free as in speech, not free as in beer" – in other words, there's much more to Free Software than its price. The fact that the English word "free" has both an economic and a political meaning causes much concern to Free Software advocates, because it confuses what are to them two very distinct ideas; in fact, many advocates prefer the Spanish "software libre" as a more precise way to describe the essential nature of their beliefs.

In the late 80's a group of these hackers organized the FSF with the stated goal of developing a complete Unix operating system clone that could be freely distributed with source code. They called their project "GNU," which (seriously) stands for "GNU is Not Unix" (the hacker sense of humor is a mysterious thing). The FSF proceeded to develop some very powerful tools that would be necessary components of a free operating system. They developed the concept of Free Software and wrote the GPL to implement it for the code they wrote.

The FSF encouraged other developers to use the GPL, and it has become the predominant license agreement in the Open Source world. Because of its importance, I'll dissect the GPL, and some of its cousins, in much greater detail later in this paper. Prior to that, it's important to understand why the GPL is what it is; it is a political as much as a legal document.

The GPL and similar licenses are designed with one key concept in mind: all users must have the freedom to modify and redistribute software licensed under its terms. You may build upon the contributions of others to the community, but you cannot hijack it. No one can create a proprietary version of a GPL'd program by distributing object code only, while keeping either the original, or the modified, source code locked up.

The Free Software Foundation has coined the term "copyleft" to describe the philosophy of the GPL: "Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. Copyleft guarantees that every user has freedom" (http://www.fsf.org/copyleft/copyleft.html).

III. A BRIEF HISTORY OF OPEN SOURCE SOFTWARE – PART 2

The GNU project remained, at least to the commercial software world, little more than a curiosity until about 1994, when two other groups almost simultaneously developed clones of the key Unix component called the "kernel" and made them available under free-software licenses. These were the "FreeBSD" and "Linux" projects. Both continue today, though Linux is by far the most popular and gathers the most publicity (and, until the crash, the most venture capital).[3]

These new kernels, coupled with the work FSF had already done on GNU, meant that hobbyists and students could put together a complete Unix-like system running on a PC computer that rivaled the power of commercial machines. And, because all this code was under the GPL or similar licenses, no one could divert it down a proprietary development path.[4]

Until recently, the thought of using Open Source software didn't even occur to most IT organizations. There was a strongly held view that the only reliable software was commercial software, and that "free" software by definition was shoddy and amateurish. The fact that there was no commercial support available for non-commercial software sealed its fate in the corporate world.

As IT folks with the hacker nature began playing with Linux and FreeBSD at home, they noticed that, contrary to received wisdom, the software was actually very stable. In fact, some would argue that these free upstarts are more reliable than most of their commercial counterparts. They also noticed that Open Source software tended to have frequent updates, rich features, and good performance. They began to question why such software wouldn't work in the business environment as well as the home and academic ones.

One of the most powerful arguments made for Open Source is that it may be the best path to high reliability. The open development process means that source code goes through a peer review and quality checking process that's far beyond the capability of any closed development shop. There are two reasons for this. First, the test process is open to the public, so the code is exercised under a broader range of conditions than a private company could create. Second, the availability of source code means that many eyes, not just a privileged few, are examining it and looking for flaws.

The combination of rapid development cycles and peer review has shown its value in the security area. Like all software, Open Source programs have bugs, and some of those bugs create security vulnerabilities. However, unlike proprietary products, which have long development cycles that discourage frequent updates, Open Source bugs are found and fixed very rapidly. And, using the Internet as a distribution channel means that updates are available virtually instantly.

Gradually, Open Source began to penetrate the business world. Internet servers were the first toehold, and today a very large proportion of web traffic is hosted on Open Source systems. In fact, as of February, 2004, 67% of the web servers on the Internet use Apache, an Open Source product (compared to 21% for Microsoft),[5] and the vast majority of all Internet email travels through Open Source mail servers. There are numerous other examples of Open Source software at the heart of the web.

Although Open Source on the desktop is not nearly as widespread as its use in in the server farm, Linux-based desktops are popping up, and the OpenOffice.org applications suite is gaining popularity as an Open Source alternative to Microsoft Office. Perhaps the biggest recent news in the battle for the desktop was a leaked IBM memo challenging its IT organization to move to a Microsoft-free infrastructure by the end of 2004.

IV. THE PHILOSOPHY OF OPEN SOURCE SOFTWARE

While the philosophy of Free Software is a political one, the philosophy of Open Source software is a practical one. In the brief history above, for practical purposes I left the impression that all non-commercial software is Open Source. That's not true. Not all free (as in beer) software is developed in an open environment, and much of it, particularly in the DOS/Windows world, is distributed only in object code form. Sometimes, source code is available, but a small group maintains tight control over development and user contributions aren't welcomed.

Open Source software is a distinct species of free-as-in-beer software characterized by development philosophy rather than license terms. It relies on development in the public eye, using the Internet as a medium to let many developers contribute to the work, and to allow testing by anyone and everyone. Actually, the distinction between "testing" and "using" Open Source programs is often blurred – Linux was widely used long before it reached the magical "version 1.0" that symbolizes a production release.

Open Source development is inherently a cooperative venture. Developers are usually volunteers, doing the work because they enjoy it, or perhaps because the product under development solves a problem that they are currently facing. A number of projects – notably the Linux kernel – have generated sufficient commercial interest that companies have assigned paid programmers to work on them, but that's still the exception rather than the rule. In an Open Source project, the group leader may serve more as a project manager than as a programmer. Depending on the origin of the project, the group leader may provide the overall architecture of the program, ensure that contributors are developing code that fits into that scheme, and make the final decision about what goes into the code base. Other projects have a more democratic focus, and these decisions may be made by committee, or even by a voting process.

The development of Linux is the prime example of the Open Source model. Originally conceived by Linus Torvalds, a Finnish computer science student, the initial versions of the Linux kernel were an outgrowth of his classwork, but he made them available on the 'net and rapidly gained a large following of programmers who, because it was a cool idea, made major contributions to the code. The current Linux kernel incorporates the work of literally thousands of developers, and Linus serves as coordinator, spokesman, and keeper of the faith, ensuring that the overall project remains true to its goal.

Certain contributors, based on their proven mastery of their subject matter, have become de facto owners of parts of the Linux kernel, and they make decisions within their sphere (subject always to Linus having the final say). Developers communicate via email lists, and a strong culture of consensus-building exists. All beta versions of the code are immediately available on the Internet. A bug reporting mechanism allows all users to report problems and suggest enhancements.

And, of course, the "official" development group is only part of the story of an Open Source project. Since the source code is always available, anyone can make their own modifications, some of which may even be improvements, and both use and distribute them. Thus, if the original development team loses interest, or no longer meets the needs of users, they hold no monopoly on the code. An offspring group can (and often has) taken over the project and moved it forward after all the original contributors are gone.

One of the leading proponents of the Open Source development model, Eric S. Raymond, has written a seminal article called "The Cathedral and the Bazaar," (www.kde.org/food/cathedral/cathedral-paper.html[6]). In it, he analyzes through the example of an actual Open Source software development project why the noisy and "promiscuous" development style typified by Linux seems to work better than the quiet crafting of elegant

systems in a closed environment (in other words, the vision, if not the reality, of typical commercial development). After noting that his relatively small project ended up with over one thousand individual contributors, he closes with the insight that:

> Perhaps in the end the Open Source culture will triumph not because cooperation is morally right or software 'hoarding' is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the closed-source world cannot win an evolutionary arms race with Open Source communities that can put orders of magnitude more skilled time into a problem."

To summarize the philosophical underpinnings of Open Source, I can't do better than quote from www.opensource.org/intro.html:

> The **basic idea behind Open Source** is very simple. When programmers on the Internet can read, redistribute, and modify the source for a piece of software, **it evolves**. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems **astonishing**.
>
> We in the Open Source community have learned that this rapid evolutionary process produces **better** software than the traditional closed model, in which only a very few programmers can see source and everybody else must blindly use an opaque block of bits. (Emphasis in original.)

V. THE COMMERCIAL BASIS OF OPEN SOURCE SOFTWARE

Linux and FreeBSD, and other lesser-known Open Source projects, were curiosities at first, but gradually people began to realize that it might be possible to make money from this unique licensing model. Prior to the dot-com crash, Linux in particular became a favorite of the investment world, and numerous companies went public to massive stock-price pops by having the word "Linux" in their names. Although the investment bubble has burst, Open Source has continued its infiltration, and it is no longer something the commercial world can ignore.

How does an Open Source company survive if it can't charge big bucks for software licenses? Most follow one of two business models:

1. Providing support services. Just because source code is available doesn't mean that you have the skill to do anything with it. Many, perhaps most, Open Source companies focus on software support, including activities such as publishing documentation, conducting training, providing technical support, and performing substantial customization to meet a customer's unique requirements.

2. Providing Linux "distributions." Linux itself isn't a complete solution. It needs to be packaged with hundreds of other programs to form a complete computing environment. Several groups, both commercial and non-profit, have developed Linux distributions (which are usually made available on CD-ROM) that include these programs as well as tools to install and manage the system. The most famous distribution is RedHat, which although currently trading at a small fraction of its IPO price, has managed to hover around profitability since 2002.

A third model, based on the idea that anyone who knew enough buzzwords could float an IPO, has proven itself to be of limited viability, and most of the companies built around it have disappeared since 2000.

Having described the environment of Open Source, I'll now go into painful detail about the primary license documents used to implement that model. I will focus first and foremost on the GNU General Public License and its rather odd but very important offspring, the "Lesser GPL," as these are the most common, the most complex, and the most troublesome of the free software licenses. I will also provide brief descriptions of the BSD License, the MIT License, and the Artistic License. Copies of all these licenses are included at the end of this paper.

VI. THE GPL

In 1988, the Free Software Foundation published the first version of the General Public License. The second, and current, version was published in 1990. The GPL has been the subject of much heated discussion among both computer lawyers and hacker-lawyer-wannabes. Much of the heat stems from the fact that the GPL isn't a particularly clearly written document, and it requires careful parsing to make sense.

Another reason for controversy is the fact that the GPL approaches the world from a perspective totally different than that of traditional licenses. The software licenses that you're familiar with almost certainly focus primarily on limiting the rights of the licensee to use the code; distribution to third parties is usually flatly prohibited in a very few unambiguous words. The GPL takes the opposite tack. It focuses exclusively on distribution, not on use. And the way distribution is handled in the GPL is likely to be unfamiliar: its focus is on encouraging, not limiting, distribution of the licensed program's source code.

Commercial entities wishing to use GPL'd software in their own products need to be aware of one thing above all others. The GPL is sometimes referred to as the "General Public Virus" because it "infects" derivative works. In simple terms, *incorporating any GPL'd code in your software makes the entire derivative work subject to the GPL*, with the corresponding requirement to provide full source code, and allow unlimited distribution, to anyone who asks,

for only the cost of reproduction. However, not every use of GPL'd code creates a derivative work. GPL opponents (including Microsoft) have tended to gloss over this fact. Use of GPL'd code together with proprietary code in a way that does not create a derivative work does not pose any copyright risk. In any event, you must think carefully and fully understand the relationship of the individual components before mixing GPL'd and proprietary code, because the end result may not be compatible with a closed-source business model.

With that preliminary warning given, let's walk through the GPL. Starting at the beginning, Section 0 (hackers count the way computers do) establishes the GPL's scope. The License applies to "any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License" ("Program" is defined as any such program or work). Thus, a mere reference to the GPL in a program's source code files or the splash screen of an object code version is enough to make it a Program subject to the GPL.

The first departure from standard software licenses also appears in Section 0 with the revelation that the GPL has nothing to do with the *use* of a Program: "[a]ctivities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted . . ."[7]

Section 1 allows unrestricted copying and distribution of the Program's source code in unmodified form. However, the distributor must "conspicuously and appropriately" include a copyright notice and disclaimer of warranty, and must keep all references to the GPL intact. Contrary to myth, the GPL doesn't require that software be given away for free. Section 1 expressly states that "You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee." On the other hand, the

reality of the GPL makes it highly unlikely that any attempt to charge a significant fee to transfer a GPL'd program will succeed: anyone else has the right to undercut your transfer fee.

In apparent contradiction of Section 0's statement that a mere reference to the GPL is enough to bring a program under its terms, Section 1 requires that a copy of the GPL be included with any distributed copy. This inconsistency can be explained by reading Section 1 as a condition of distribution, not as a limit on the GPL's applicability. In other words, failure to include the GPL with a distributed copy is a violation of Section 1, but does not remove the Program from the GPL's coverage.

Section 2 addresses the right to modify the Program and is in many ways the heart of the GPL. Because the most practical way to modify a computer program is to work with the human-readable source code, this section focuses on source rather than object code. Section 2 begins with a simple and straightforward declaration: "You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or wok under the terms of Section 1 above . . ." However, this right is subject to compliance with three conditions.

The first and third conditions are fairly straightforward. Section 2(a) requires modified files to carry a prominent notice stating that they were changed. Section 2(c) requires Programs (in most cases) to display a "splash screen" on startup that includes a copyright notice, a disclaimer of warranty, a statement that the Program may be redistributed under the GPL, and finally, instructions on how to view the GPL.

The second condition, Section 2(b), is more substantive. It makes the GPL incompatible with a closed-source environment by requiring any work that "in whole or in part contains or is derived from the Program or any part thereof" to be licensed "as a whole at no charge to all third parties under the terms of this License." This clause has two important consequences. First, the

entire modified work must be distributed under the GPL; any new content added to the original work is thus automatically free software; this is the source of the "GPL as virus" concept. Second, no one may charge a license fee (as distinct from a distribution charge) for access to software that is derived from GPL'd code. The purpose of the GPL, implemented in Section 2 (b), is to prevent a modified Program from being "taken proprietary" or made subject to license fees.

The remainder of Section 2 elaborates on this requirement and creates a very limited exception to its breathtaking implications. The exception is contained in the second full paragraph, which states that although the three conditions on distribution apply to the work as a whole, "[i]f identifiable sections of [the] work are not derived from the Program, and can reasonably be considered separate and independent works in themselves," those sections may be *separately distributed* free of the GPL. However, if they are distributed together with GPL'd code in a "derivative or collective work based on the Program," the GPL will apply to the whole work.

It's fair to ask what this means in the real world. Here is an example. Graphics hardware manufacturer VideoCo decides to develop a driver module that lets its products work with Linux. If the module is developed completely from scratch and does not contain any GPL'd code, it could be distributed by itself as a stand-alone product under whatever license terms VideoCo chooses.[8] But if VideoCo decides it would its hardware sales would benefit if its driver module was included as part of the Linux kernel (which is distributed under the GPL), it may only do so by putting the module under the GPL and making its source code freely available. The GPL's intent is to ensure that source code is available for all the pieces that touch GPL'd Programs. The price of withholding source code is exclusion from the mainstream of free software distribution.

In a further attempt to clarify, Section 2 goes on to state:

> Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works.
>
> In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

This makes it clear that the mere presence of a GPL'd Program doesn't infect an entire hard disk or CD-ROM.

Questions about mixing GPL'd and proprietary code at the "compilation" level are very common. To avoid any possibility of a compilation claim, it would be wise to clearly separate the two species of software on any CD or other distribution medium. If feasible, it would be best to use two separate directory structures, one for GPL'd and the other for proprietary components. The installation process should be designed so that the GPL'd and proprietary components are installed in separate and visible steps.

Section 3 addresses object code versions of the Program and allows their distribution provided that any one of three conditions is met:

1. the object code is accompanied by a source code distribution that complies with Sections 1 and 2;

2. the distribution includes a written offer, valid for at least three years, to give the complete source code to any third party for no more than the cost of physically performing the distribution (*i.e.*, copying and mailing costs); or

3. for non-commercial distribution of object code versions, the distribution includes the information about receiving the source code that was received from the next person up the distribution chain (in other words, users may pass on object code versions to their friends if they retain the notices contained in the distribution they received).

The important result of Section 3 is that there is no such thing as "object code only" distribution in the GPL world. The right to modify software, guaranteed by the GPL, is only meaningful if source code is available, and to achieve this end the GPL makes it impossible to lock up the source code of programs under its protection.

Sections 2 and 3 form the heart of the GPL, and their requirements are the ones you will need to consider most carefully when advising a client about distributing GPL'd code. Some of the other GPL provisions are also quite surprising, and are worth careful review.

Section 5 makes the GPL operatively a shrinkwrap license while denying that it is a contract of adhesion. It also reemphasizes the License's focus on modification and distribution, but not use:

> You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing, or modifying the Program or works based on the Program.

Section 6 states that the recipient of each redistribution of the Program automatically receives a license from the original licensor to copy, distribute, or modify the Program, and that the Licensor may not impose any further restrictions on these rights. Thus, subsequent distributors cannot weaken the GPL by adding further restrictions. Section 6 also lets licensors breathe a bit easier by stating that they are not required to enforce third parties' compliance with the GPL.

Section 7 attempts to avoid limitations on the right to distribute (such as requiring royalty payments) that might result from infringement claims or the like. In short, if a court requires conditions on distribution that are inconsistent with the GPL, the right to distribute is simply terminated; the GPL's philosophy is that no distribution is better than restricted distribution.

Section 8 furthers this by allowing the original licensor to prohibit distribution of the Program in countries that would impose restrictions based on patents or copyrighted interfaces.

Sections 11 and 12 include fairly standard warranty disclaimers and limitations on liability. These protections extend, to the extent permitted by law, to "all copyright holders, or any other party who may modify and/or redistribute the Program."

In summary, the GPL's goal is to ensure that users are free to modify and further distribute GPL'd Programs subject only to the requirement that they pass that freedom on to all subsequent users. The danger is that incautious use of GPL'd code can result in an application becoming free software without the copyright holder having any say in the matter.

VII. THE LESSER GPL

The FSF realized that there was one circumstance under which the GPL might be considered a bit presumptuous even by those fully supporting the concept of free software. That case is where an otherwise new and original work ends up incorporating (or "linking") code from a pre-existing standard "library" of software routines into the run-time version. The use of libraries providing the code to implement commonly used functions is fundamental to modern computer programming; it avoids repetitive work and encourages portability of code from one environment to another. The compiler programs that are used to turn source code into executable programs typically include several libraries of standard functions for linking purposes, and Unix-based environments like Linux include a standard set of libraries upon which virtually all programs rely.

The catch is that if a GPL'd library is used, the resulting combination of object code and library is a derivative work subject to the GPL and all its requirements for free source code distribution. In other words, a program developed completely independently but compiled using GPL'd standard libraries becomes subject to the GPL by the mere fact of being linked with those libraries. That is a result that could be charitably viewed as over-reaching.

The Lesser GPL[9] is important because it (albeit grudgingly) recognizes that this is not always a desirable outcome and adjusts distribution requirements for programs linking to LGPL'd libraries accordingly. It acknowledges that use of these libraries does not impact the resulting executable program. This is an important issue because the GNU C compiler ("gcc") and its accompanying LGPL'd libraries are very commonly used in both open and closed source development projects. Unfortunately, the LGPL is even more difficult to parse than its cousin, and subtle technical distinctions can have major legal ramifications.

The LGPL is a modified version of the GPL with the addition of language addressing how licensed libraries may be used. Section 0 defines a "library" as "a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables." For technical reasons that may become apparent below, the term "executables" is used in the LGPL instead of the very similar concept of "object code" used in the GPL.

The LGPL creates an important distinction between a "work based on the Library," which contains code derived from the library (for example, a modified version of the library), and a "work that uses the Library," which must be combined with the Library in order to run (for example, a program that uses functions provided by the Library at compile-time).

This distinction is critical. Under LGPL Section 2, a "work based on the Library," *i.e.*, a modified version of the Library, is subject to essentially the same source code availability requirements as the GPL, with some additional requirements to maintain compatibility between the original and modified versions. Because modification and distribution of libraries in source code form is a situation that will arise but seldom in commercial settings, I won't talk further in this paper about "works based on the Library."

Section 5 defines a "work that uses the Library" more specifically as "a program that contains no derivative of any portion of the Library, but is intended to work with the Library by being compiled or linked with it," and says that "[s]uch a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License." Although this sounds as though it might give such programs a way out of the LGPL structure, the story is not quite so simple. Section 5 goes on to state that this exemption applies only to certain, usually source code, versions of the program:

> However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the

> Library), rather than a "work that uses the Library." The executable is therefore covered by this License. Section 6 states the terms for distribution of such executables.

At bottom, this somewhat circular language means that source or object code versions that merely reference functions contained in the Library do not fall under the Lesser GPL, while executable versions that directly incorporate Library code to implement such functions, do.

So, what does Section 6 allow us to do with such executables? Not surprisingly, this is a lengthy and complex provision. In short, executables that include code from the Library may be distributed under terms of the licensor's choosing, "provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications." This is a significant proviso, and it is likely to conflict with standard software license terms for end users; as a result, the Lesser GPL does not totally solve the problem of using GPL tools in closed-source projects.

Section 6 imposes further requirements largely aimed at implementing the licensee's ability to modify the executable. In addition to establishing copyright and other notice provisions tracking those in the GPL, it requires the licensor make available the source code and tools necessary to permit the user to modify the Library (but not necessarily the independently-developed work) and then re-link to create a new executable program. Five ways to do this are offered, with two primary results:

1. If the program is "statically linked," the licensor must make available the full Library in source code form and also make the work using the Library available separately in either object or source code form.[10] Static linking means that the Library is permanently linked into the executable program during the compilation process. This is common in the DOS world, but less so in Unix-like environments. In this case, the licensor must provide all the tools (other than those that normally come with the operating system environment) necessary to reproduce the

executable from the combination of the object code version and the Library. A significant commercial concern is that if those tools are themselves proprietary, the licensor theoretically has to acquire a license for any user that requests one, and cannot pass through any license fees it pays to do so.

2. If the program is dynamically linked to a shared library that already exists on the computer (in other words, linking occurs when the program is run, not when it is compiled), no further action is necessary. Most Linux and FreeBSD systems primarily use shared libraries, so this option makes the LGPL work well in those environments. This option also applies to "DLL" libraries in the Windows world.

In summary, the Lesser GPL allows libraries adopting it to be fairly easily used in shared-library environments such as FreeBSD, Linux, and possibly Windows. It does not work as well (from the closed-source developer's point of view) in "static library" environments, because it requires distributing intermediate files and development tools that are not normally given to end users. It does, however, avoid a closed-source shop's biggest concern by not requiring source code distribution.

VIII. THE BSD LICENSE

Significant portions of modern Unix implementations (both free and commercial) derive from work done at the University of California – Berkeley. The original Unix work there was called the Berkeley Software Distribution or "BSD" for short. The software was made freely available by the University under what became known as the BSD License. The BSD License's terms are short and straightforward (particularly in comparison with the GPL) and this license is used for many free software projects, including the FreeBSD Unix clone.

The BSD License simply states that "redistribution and use in source and binary forms, with or without modification, are permitted." It imposes a few non-painful conditions: retention

of copyright notices and license terms on redistribution; an agreement not to use the copyright holder's name without consent; a disclaimer of all warranties; and a complete exclusion of all liability on the part of the copyright holders and contributors. Unlike the GPL, the BSD License does not attempt to enforce freedom by infecting derivatives or requiring source code availability.

An earlier version of the BSD license included a clause requiring that credit be given to the copyright holder; the FSF opined that this "advertising" clause was inconsistent with the GPL, and that the BSD license was therefore not compatible. The advertising clause was removed several years ago, and current versions of the BSD license are deemed to be GPL-compatible.

IX. THE MIT LICENSE

Another significant component of modern Unix distributions is the X Windows System, which provides users with a graphical interface. X was originally developed at MIT, and is licensed under terms very similar to the revised BSD License (the one that does not include the advertising clause), but with a few more words defining the permitted uses, and a few less devoted to the liability limitation clause. The practical effect is that the MIT License and the revised BSD License are virtually indistinguishable.

X. THE ARTISTIC LICENSE

The Artistic License, under which Perl (an important Unix tool) and some other programs are licensed, attempts to be a simpler version of the GPL. It permits distribution of modified source code versions so long as the modifications are placed in the public domain or otherwise made freely available, or the executable programs are renamed so they do not conflict with the standard versions (which must be included in the modified distribution).

Object code versions may be distributed provided that the modified version are given non-standard names, the original object or source code versions are included in the package, and the differences between the original and modified versions are clearly documented.

The Artistic License permits the licensor to charge for distribution or support, but not for the software itself. Importantly (because Perl is a widely-used program development tool), scripts and library files provided as input to, or output from, the program do not fall under the License. In other words, a program developed using a tool licensed under the Artistic License does not fall under the License's terms, and may be licensed in whatever manner the developer wishes.

The original version of the Artistic License contains ambiguities which prevented the FSF from deeming it GPL-compatible. A revised version resolves these issues. Also, note that Perl language is actually licensed under the user's choice of either the Artistic License, or the GPL.

XI. CONCLUSION

Free software and the Open Source movement can no longer be ignored by the commercial software world. No one other than a few zealots expect that all software will adopt the Open Source model, but it's clear that free software will be finding its way into many development organizations. If you represent an organization that uses a traditional closed-source development model, it's vitally important that you advise your clients about the significance of the GPL and other free software licenses, and help them avoid development models that could result in the unintentional creation of new free software.

END NOTES

[1] It took a long time for that idea to die out. The hardware reference manual for the original IBM PC, published in 1981, included a full source code listing of the computer's BIOS system. Shortly thereafter, IBM had a change of heart and spent the next decade or more threatening companies that developed BIOSes that were a bit too similar to IBM's.

[2] The FSF web site (http://www.fsf.org/licenses/license-list.html) lists licenses which are "GPL-compatible free software licenses," "GPL-incompatible free software licenses," and "non-free software licenses."

[3] In recent months, the Open Source world has been abuzz about the claims made by The SCO Group that Linux infringes copyrights on Unix that SCO now holds. That case is so complicated, and things are moving so rapidly, that it's hopeless to try to explain it here. If you're interested in the bizarre twists and turns of SCO vs. The World, Groklaw (http://www.groklaw.net) is the one and only place you need to go.

[4] Another reason that it's difficult to co-opt Open Source projects is that most of them are the creation of so many authors that it would be virtually impossible to obtain suitable licenses from everyone who owns a piece of the copyright.

[5] Source: Netcraft Web Server Survey (http://www.netcraft.com).

[6] I hope this link works; the official URL for the paper seems to have disappeared.

[7] Since activities other than copying, distribution, and modification are outside the scope of the GPL, could the right to run a Program be restricted by a separate "use" license that operates alongside the GPL? Could that license require a fee? The GPL does not give a clear answer to these questions.

[8] One might argue that the act of loading the module into memory along with the computer creates a derivative work, but that is the act of the user, not the module's author. The question is whether the module author would then be subject to a claim of contributory infringement under the GPL; it muddies the waters further to consider that someday someone might write a non-GPL but Linux-compatible kernel with which the module could be used.

[9] The Lesser GPL was formerly known as the Library GPL; the name was changed because the FSF thought the previous name provided too much encouragement to use this license for libraries rather than the original GPL.

[10] Although not commonly done, most compilers allow the "intermediate" object code files that exist after compilation, but before linking, to be saved. These files can be distributed to comply with Section 6 because they allow relinking with modified Libraries to create a new executable.