UIC John Marshall Journal of Information Technology & Privacy Law

Volume 27
Issue 3 *Journal of Computer & Information Law - Spring 2010*

Article 3

Spring 2010

The Cathedral and the Bizarre: An Examination of the "Viral" Aspects of the GPL, 27 J. Marshall J. Computer & Info. L. 349 (2010)

Michael F. Morgan

THE CATHEDRAL AND THE BIZARRE: AN EXAMINATION OF THE "VIRAL" ASPECTS OF THE GPL

MICHAEL F. MORGAN*

"For it is possible long study may encrease, and confirm erroneous Sentences: and where men build on false grounds, the more they build, the greater is the ruine."[1]

I. INTRODUCTION

The use of open source software in commercial products has become commonplace during the last ten years. At first, open source software was used primarily for internal deployments within universities and corporate IT departments. These uses demonstrated the quality and usefulness of many software packages developed using the open source methodology. With the growth of the Internet, high-quality,[2] feature-rich open source software has been used for increasingly important applications. For example, the most popular web server is and has long been the Apache HTTP server.[3] Various surveys have indicated that open source programs are the leading products in the email server,[4] DNS

* Michael F. Morgan. LaBarge Weinstein Professional Corporation, 515 Legget Drive, Ottawa, Ontario, Canada K2K 3G4. The opinions expressed are those of the author only.

1. THOMAS HOBBES, LEVIATHAN 317 (Penguin Books 1985) (1651).

2. *See generally* ERIC S. RAYMOND, THE CATHEDRAL & THE BAZAAR: MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY (2001). Raymond characterizes the quality advantages of open source software as follows: "Given enough eyeballs, all bugs are shallow." *Id*. at 30. This formulation is now known as Linus's law, after Linus Torvalds the primary developer of the Linux kernel. *Id*.

3. David A. Wheeler, Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the numbers!, ch.2 §1, (Feb. 16, 2007), http://www.dwheeler.com/oss_fs_why.html#market_share (stating that Netcraft surveys have consistently shown the Apache server as the number one web server since it first assumed that rank in April 1996).

4. *Id.* at ch.2 §22 (citing a survey by D. J. Berstein in 2001 that found the top three programs in this category were unix sendmail (42%), Microsoft Exchange (18%), and qmail (17%)). A Message Transfer Agent (MTA) is a program used to deliver email. Both sendmail and qmail are available in source code, although qmail is generally not consid-

server,[5] and reverse domain lookup[6] categories. The successful use of open source software in these applications drew the attention of commercial software vendors. As a result, it is becoming more difficult to find commercial software products that do not use open source software to some extent. Because of increasing competition, including competition from low-cost, offshore suppliers and the increased use of low-cost offshore developers, the use of large amounts of high-quality, royalty-free software is seen as significant commercial advantage. Aside from the obvious cost and time-to-market advantages, the use of open source software can also allow commercial software vendors to take advantage of widely-used standardized components, thereby allowing them to focus their limited resources on value-added competitive differentiators.

Invariably, if a commercial software vendor decides to use open source software, that vendor is going to have to decide whether to use software licensed under the GNU General Public License (the "GPL"). While the use of GPL-licensed software for internal deployments is reasonably well understood, the use of GPL-licensed software with commercially distributed software packages is much less well understood and is the subject of many conflicting views. The most contentious issue related to the use of GPL-licensed software in commercial contexts is the application of the so-called "viral" provisions of the GPL. These viral provisions are purported to make distribution of a GPL-licensed program contingent on a requirement to "license the entire work, as a whole, under [the GPL] to anyone who comes into possession of a copy."[7] The

---

ered to be open source because its author has imposed restrictions on the distribution of qmail modifications.

5. *Id*. at ch.2 §25 (citing a survey by Don Moore that found the "bind" DNS server was used by approximately 70% of domains surveyed; commercial offerings in this category were used in only about 25% of domains surveyed). A domain name server (DNS) is software that takes human readable names like google.com and translates them to their corresponding numeric IP address. For example, the IP address for google.com is 216.239.39.99.

6. *Id.* at ch.2 §24 (citing a survey by Bill Manning in 2000 that found 95% of all reverse domain resolution servers were using a variant of the open source program "bind"). A reverse domain resolution server looks up an IP address to obtain a domain name. For example, a reverse domain name lookup on 216.239.39.99 would return the domain name google.com.

7. GNU Operating System, *GNU General Public License, Version 3*, §5(c), June 29, 2007, http://www.gnu.org/licenses/gpl-3.0.txt. Version 2 of the GPL contains a similar requirement which provides that any "work based on the program also be licensed under the GPL." GNU Operating System, *GNU General Public License, Version 2*, §5(c), June 1991, http://www.gnu.org/licenses/gpl-2.0.txt. One of the problems with the GPL is the variety of terms used to characterize the "viral" provisions. For example, the GPL.v2 also refers to "modifications", "the Program or any derivative work under copyright law," "a work containing the Program or a portion of it," "any work . . . that in whole or in part contains or is derived from the Program or any part thereof,' "identifiable sections of [a] work [that] are not derived from the Program, and can be reasonably considered independent and separate

use and distribution of GPL-licensed software with commercial software can potentially oblige a commercial software vendor to license its proprietary source code under the terms of the GPL to persons who have received the corresponding binary product. Once that source code has been licensed to someone under the GPL, the person or entity receiving the source code will be entitled to make that source code available over the Internet at no-charge for unlimited uses. The detrimental effect on the market for an affected commercial software product is obvious.

Historically, the GPL has been a difficult document to understand. Various commentators have described the GPL as a mixture of a legal contract and an ideological manifesto,[8] the constitution of open source,[9] and "a threat to the intellectual property of any organization making use of it."[10] Part of the difficulty in understanding the GPL is that earlier versions of the license were drafted in a very casual style. The confusion caused by this casual drafting is reflected in the varied and conflicting opinions about the viral effects of the GPL.[11] Unfortunately, while much

---

works in themselves," "derivative or collective works based on the Program," "work[s] written entirely by you," and "mere aggregation[s]." *Id.*

8. Andrés Guadamuz González, *Viral Contracts or Unenforceable Documents? Contractual Validity of Copyleft Licenses*, 2004 EUROPEAN INTELL. PROP. REV. 331, 333 (2004), *available at* http://opensource.mit.edu/papers/guadamuz.pdf.

9. ROD DIXON, OPEN SOURCE SOFTWARE LAW 30 (2004).

10. Craig Mundie, Senior Vice President, Microsoft Corporation, Prepared Text of Remarks at the New York University Stern School of Business, (May 3, 2001), *available at* http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.mspx.

11. Chris Nadan, *Risks Associated with Open-Source Licensing and Usage*, 19 COMPUTER L. ASSOC. BULL. 53, 56 (2004):

> The most pernicious urban legend is that dynamic linking is not viral under GPL. The GPL is rather ambiguous about whether proprietary code dynamically linked to GPL code becomes contaminated (dynamic linking is where the interaction between GPL code and non-GPL code occurs only at runtime.) . . . There is a hearty debate in the open-source community about the contaminating effect of dynamic linking. The open source community typically claims that such linking does not contaminate. The question I would pose to you, as a legal advisor, is: Maybe they're right, maybe they're not. But can your client afford to be the test case? *Id.*

Jason B. Wacha, *Open Source, Free Software, and the General Public License*, 20 COMPUTER & INTERNET LAW 20, 22-23 (Mar. 2003):

> In determining whether an application can remain proprietary, the licensing status of the library to which the application is linked is of key importance. 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. (footnotes omitted).
>
> . . ..
>
> [T]here 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. *Id.*

LAWRENCE ROSEN, OPEN SOURCE LICENSING: SOFTWARE FREEDOM AND INTELLECTUAL PROPERTY LAW 121-22 (2005) (stating that "the legal analysis of what constitutes a derivative

of the language in the GPL has been improved in version 3 (the "GPL.v3"), these changes do not appear to have clarified the viral provisions.[12] Despite the lingering confusion about the potential viral effects

---

work simply doesn't depend on the style or mechanism of inter-program linking."). *Id.* at 287. "Nothing in the law of copyright suggests that linking between programs is a determinative factor in derivative work analyses by courts—except perhaps as evidence of one of the abstract, nonliteral, copyrightable aspects of the software, such as program architecture, structure, and organization." *Id.* "It is highly doubtful that the dynamic linking by a proprietary application to a GPL'ed library would normally result in a derivative work of the original licensor." DIXON, *supra* note 9 at 32-33:

> The OSI interpretation suggests that a completely separate proprietary program will become GPL if it merely shares data with GPL code, even if the only such sharing occurs while the program is actually running. This phenomenon—that proprietary code becomes open source whenever combined with open source code—is the so-called "viral effect" of the copyleft licenses. Like a virus, GPL code infects any proprietary code with which it is combined, turning it into GPL code as well. (footnotes omitted). Nadan, *supra* note 11, at 360.

In annotation #9 to the Open Source Definition, the Open Source Initiative (the "OSI") said that "the GPL is conformant with this requirement. GPLed libraries 'contaminate' only software to which they will actively be linked at runtime, not software with which they are merely distributed." Open Source Initiative, Open Source Definition, Version 1.8, http://web.archive.org/web/20010330062316/www.opensource.org/docs/definition.html (last visited Apr., 26, 2009). Subsequently, the OSI amended this annotation to state "the GPL is conformant with this requirement. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed." Open Source Initiative, Open Source Definition, Version 1.9, http://www.opensource.org/docs/definition.php (last visited Apr. 26, 2009):

> The FSF's position that the GPL is "a license not a contract" probably means that the FSF cannot successfully seek injunctive relief to force anyone to lay open-source [sic] code. The fact that it does not charge license fees for its software probably means it cannot seek damages based on the amount of a reasonable royalty — though it could still rely on statutory damages. So the FSF's position — comply or stop using our stuff — is, quite neatly, mostly what they could get under the law. Heather J. Meeker, *Open Source and the Legend of Linksys*, LINUS NEWS:TECH BUZZ, June 28, 2005, http://www.linuxinsider.com/story/43996.html?wlc=1240785572.

12. Jonathan Schwartz, *The Participation Age,* JONATHAN'S BLOG, Apr. 28, 2005, http://blogs.sun.com/jonathan/date/20050404:

> [T]he GPL expressly limits choice by disallowing the inclusion of non-GPL code into GPL projects - and exports a form of IP colonialism to nations seeking to create their own means of production.
>
> . . . .
>
> And having just spent some time with a breadth of network equipment OEM's at the GSM World Congress; and a series of representatives from developing nations at a recent customer event – I can assure you they are both suspicious of richly valued companies, with enormous patent portfolios and legal teams, evangelizing the benefits of the GPL and the elimination of software patents. They're beginning to see it as a means of forcing them to disgorge their intellectual property, and convey it to those same richly valued companies. Free software they understand: they also understand exploitation).

Richard Stallman, President, Free Software Foundation, Transcript of Richard M Stallman's speech at New York University (May 29, 2000), *available at* http://www.gnu.org/events/rms-nyu-2001-transcript.html:

of the GPL, it remains the license of choice for a majority of "open source"[13] projects. Furthermore, there is no reason to believe this situation is going to change any time soon.

Given the uncertainty about GPL viral effects and the potentially serious consequences for a commercial product, one would expect that GPL-licensed software is never used with commercial products. However, this is not the case.[14] GPL-licensed software can find its way into commercial products in a number of ways. Perhaps the most common way is through simple inadvertence. A software designer faced with an impending deadline and a requirement for a large but competitively unimportant piece of functionality may after searching the Internet find a software package that provides exactly the functionality he or she requires, and, even better, this software has been tested and appears to be bug-free. The license, to the extent it is even examined, is confusing but talks at length about being free. With a release deadline approaching, a designer may simply decide to use the GPL-licensed software. The designer's problem is solved, but his or her employer's problems may be just beginning.

Another way in which GPL-licensed software can be used in a commercial product is through a reasoned (and likely difficult) decision. The viral provisions of the GPL will only apply to another software program if that program is "combined" with GPL-licensed software in certain ways. The Free Software Foundation (the "FSF") has stated that certain types of program-to-program interactions will not engage the viral provisions of the GPL. Accordingly, there are at least some situations in which GPL-licensed software can be used safely with commercial software. However, the various ways in which programs interact, the

---

> The freedoms to change and redistribute this program become inalienable rights—a concept from the Declaration of Independence. Rights that we make sure can't be taken away from you. And, of course, the specific license that embodies the idea of copyleft is the GNU General Public License, a controversial license because it actually has the strength to say no to people who would be parasites on our community.

13. The term "open source" is used here simply to describe a software development project that makes source code available to users. For a discussion of the differences between the "open source" movement and the "free software" movement, *see* Eric S. Raymond, *The Revenge of the Hackers*, *in* OPENSOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 207 (Chris DiBona et al. eds., 1999), and Richard Stallman, *The GNU Operating System and the Free Software Movement*, *in* OPENSOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 53 (Chris DiBona et al. eds., 1999).

14. *See*, *e.g.*, Daniel Lyons, *Linux's Hitmen*, FORBES.COM, Oct. 14, 2003, http://www.forbes.com/2003/10/14/cz_dl_1014linksys_print.html; Progress Software Corp. v. MySQL AB, 195 F. Supp. 2d 328 (D. Mass 2002); Welte v. Sitecom Deutschland GmbH, No. 21 0 6123/04 (Landgericht Muenchen I) (May 19, 2004); District Court of Munich, 12 July 2007, case 7 O 5245/07 (*Welte v. Skype Technologies S.A.*), *available at* http://www.ifross.de/Fremdartikel/LGMuenchenUrteil.pdf.

copyright law relevant to these interactions, and the scope of copyright protection for APIs, data structures, client-server protocols and other components of a typical software package make it difficult to determine whether a particular interaction is safe.

OUTLINE

While there is a growing body of literature dealing with the GPL, the potential viral effects of the GPL do not appear to have been analyzed in a detailed technical manner. This paper will attempt to demonstrate that a proper legal analysis of the viral effects of the GPL is dependent on a detailed technical understanding of the specific mechanisms used for each type of program-to-program interaction. Once these technical mechanisms are properly understood it will then be possible to identify the applicable copyright law needed to assess the viral effects of the GPL.

The technical analysis in this article will consist of an examination of the three main ways in which computer programs interact: static linking, dynamic linking, and inter-process communication. Section 2 of this paper will provide a detailed technical description of these types of program-to-program interactions. Subsection 2.A will start with a relatively high-level description of program linking. This subsection will explain why programs are linked, describe the various tools used for linking, and provide a basic description of linking. The next two subsections, 2.B and 2.C, will provide a detailed description of static and dynamic linking in a typical UNIX environment.[15] Subsection 2.D will conclude the technical analysis with a detailed description of inter-process communication.

Section 3 will build on the technical information provided in Section 2 with the goal of identifying the areas of copyright law relevant to an analysis of the viral effects of the GPL. This analysis will start in Subsection 3.A with an examination of those parts of the GPL that have been characterized as viral. This examination will show that the derivative works right is crucial to understanding the viral effects of the GPL. In Subsection 3.B, the derivative works right and related jurisprudence will be examined. Subsection 3.C will use the derivative works jurisprudence to assess the viral effects of the GPL for statically linking. This analysis will conclude that basic copyright principles can be used to reach definitive conclusions about the viral effects of the GPL when statically linking. In Subsection 3.D, the viral effects of the GPL for dynamic linking will be examined. Unlike the case with static linking, an analysis of dy-

15. There are some technical differences between linking mechanisms and formats across computing environments. However, these differences are unlikely to have any copyright significance. For simplicity, this paper will consider the ELF file format as used on a UNIX operating system.

namic linking will require an understanding of a number of complex areas of copyright law. In particular, the scope of copyright protection for technical interfaces will need to be examined, which, in turn, will require an examination of copyright protection for methods of operation and data structures. Finally, Subsection 3.E will examine GPL viral effects in inter-process communication. As in the case of dynamic linking, copyright law related to technical interfaces and data structures will be crucial in understanding the viral effects of the GPL in inter-process communication. In each of Subsections 3.C, 3.D and 3.E, the expected viral effects of the GPL will be described. In areas where the law is unsettled, the open questions will be identified and suggestions will be made about how a court might answer these questions. In each case, expected or projected viral effects of the GPL will be compared to statements made by the FSF.

II. PROGRAM-TO-PROGRAM INTERACTIONS

Software is generally understood to exist in two formats: source code and object code.[16] Source code can be read and understood by humans and is the preferred format for programming. Object code can be read and processed by computers and is the format used by computer hardware when executing a program. The existing jurisprudence and legal literature have used these traditional classifications because this level of detail has generally been sufficient for assessing copyright issues to date. However, when assessing GPL viral effects, and in particular when trying to determine whether one computer program is a derivative work of another, it becomes necessary to examine all software formats in much greater detail. Specifically, various intermediate software formats will need to be examined to fully understand the copyright consequences of certain steps in static and dynamic linking.

A. Linking

Many programs contain hundreds of thousands and sometimes millions of lines of source code. These programs are usually developed and worked on continuously by multi-person design teams. Consequently, it is generally not practical to keep the source code for a program in a single file. As a result, most programs consist of multiple files called modules. Organizing a program in this way provides a number of benefits. For example, multiple programmers can work on different modules within the same program without having to worry about overwriting each other's changes. Additionally, changes to one module will generally

16. This description ignores interpreted languages. As discussed later, a classification into only source code and object code form is a simplification since compilation involves the creation of a number of intermediate file formats between the source and object code formats.

not necessitate changes to other modules.[17] Finally, the segregation of various discrete functions within separate modules will generally result in stronger, more robust architectures.[18] Because computer programs are almost always developed as a collection of separate modules, a mechanism is needed to combine these modules to create a final product.

Linking is the process of gathering and combining various code and data items into a single file that can be loaded into computer memory and executed.[19] Instead of requiring every application to consist of one large monolithic file, linking allows software to be developed in multiple files. This approach to software development has numerous advantages. Among these advantages is the ability to modify and separately compile portions of a large program. Instead of having to go through the time consuming task of recompiling all of the files in a large program, linking allows a developer to re-compile only modules that have been changed. These modules can be then re-linked with the unmodified portions of the program, saving significant time and effort. In return for the advantages provided by linking, there is added complexity in the object file structure. However, this complexity is generally hidden from programmers and handled by compilers, linkers and loaders. Currently, there are two main linking techniques. The first technique is called static linking. Static linking is generally understood to mean linking at program-development time so that the resulting object code program can be loaded and executed without any further linking-related activity. The second technique is called dynamic linking. Dynamic linking is understood to mean a process in which a portion of the linking is done at development-time, but most of the linking is done at program load or run-time.[20]

Most current compilation systems provide software developers with a compiler driver.[21] This tool will run a language preprocessor, compiler, assembler and linker that collectively convert a high-level source code program into an executable program.[22] A well-known example of this type of program for the GNU compilation environment is *gcc*. The *gcc*

17. However, some changes, such as a change to a definition of a widely used data structure can create a need to recompile other modules using that data structure.

18. Organizing modules in this way allows data hiding and will force other parts of a program to interact with a particular module in a defined and regulated manner. Programs developed in this way are generally of higher quality because of the reduced chance of defects due to unexpected or unpermitted interactions.

19. RANDEL E. BRYANT & DAVID R. O'HALLARON, COMPUTER SYSTEMS: A PROGRAMMER'S PERSPECTIVE 540 (2003).

20. *Id.* at 567. All linking that can be done at development-time based on the information then available is performed. In addition, the data structures and bookkeeping information needed to allow the remaining load and run-time linking to be performed are also created and added to the object file.

21. *Id.* at 541.

22. *Id.*

compiler includes a C preprocessor (*cpp*), a C compiler (*cc1*), an assembler (*as*) and a linker program (*ld*).[23] To help illustrate some of the functions performed by these tools and to better describe the structure of an object code file, the internal programming statements needed to allow a source code file to be statically linked to a GPL-licensed library need to be described.

At the source code level, calling a GPL-licensed program from a non GPL-licensed program requires the inclusion of a reference to the name of the GPL-licensed program. The compiler driver uses this reference to locate the program being called. In the C and C++ programming languages, this reference is made using an *include* statement.[24] An actual call to a specific routine within a GPL-licensed program or a reference to a specific global variable within a GPL-licensed program is done using the symbolic name for that routine or variable. The use of *include* statements, procedure calls and variable references is the same for both static and dynamic linking.

B. Static Linking

As described earlier, static linking is a process by which code and data are combined at build-time to create a completely linked executable file. This means the final executable program is fully linked prior to execution by an end user. A static linker such as the UNIX *ld* program takes as input a collection of relocatable object files and various command line arguments and generates a fully linked executable object code file that can be loaded and run without any further linking.[25] To understand this process, one needs to understand the various types of object files, their functions, and their formats. In a UNIX environment there are three types of object files: relocatable object files; executable object files; and shared object files.[26] Relocatable object files contain binary code and data in a form that can be combined with other relocatable object files at compile-time to create an executable object file.[27] Executable object files contain binary code and data in a form that can be copied directly into computer memory and executed.[28] Shared object files are a special type of relocatable object file that can be loaded into computer memory and dynamically linked either at load-time or at run-time.[29] Shared object files will be discussed in more detail in Section 2.C.

---

23. *Id.* at 541-42.

24. In the Java programming language this function is performed by an "import" statement.

25. *Id.* at 542.

26. Bryant & David R. O'Hallaron, *supra* note 19, at 543.

27. *Id.*

28. *Id.*

29. *Id.*

At a high level, a typical relocatable object file is simply a sequence of bytes. Some of these bytes represent program code, other bytes represent program data, and still other bytes represent data used by the linker and loader to modify the program and its data so the program can be executed on a particular computer and operating system. The two main tasks performed by linker and loader systems are symbol resolution and relocation.[30] When a program is built from multiple modules, references from one module to another are made using symbols.[31] These symbols can represent various program constructs such as procedures, functions, and variables. The purpose of symbol resolution is to associate each symbol reference with exactly one symbol definition.[32] The other main task of a linker is relocation. Relocation is a process in which various memory references are adjusted so they refer to the actual location where a particular program component will be when the program is loaded into computer memory.[33] This process is challenging because when assembly is done the assembler does not know where a program will be located in memory.[34] Compilers and assemblers usually generate code and data blocks assuming a program will be loaded starting at address zero.[35] This assumption is almost always incorrect; therefore, all symbol references must be revised prior to execution so they point to the proper execution-time memory location.[36] These corrections are done using various structural characteristics of a relocatable object file; hence, it is important to understand that structure in order to understand the relocation process.

The following is the format of an Executable and Linking Format ("ELF") relocatable object code file:[37]

30. *Id.* at 575.

31. Even when references are made from one part of a subprogram to another part of that subprogram, those references are made with symbols.

32. BRYANT & O'HALLARON, *supra* note 19, at 548.

33. JOHN R. LEVINE, LINKERS & LOADERS 149.

34. BRYANT & O'HALLARON, *supra* note 19, 558.

35. LEVINE, *supra* note 33, at 5.

36. *Id.*

37. BRYANT & O'HALLARON, *supra* note 19, at 544 (2003). This chart is reprinted in its entirety from Computer Systems. *Id.* ELF is a popular format used in the Linux and BSD variants of UNIX. *Id.*

ELF header
.text
.rodata
.data
.bss
.symtab
.rel.text
.rel.data
.debug
.line
.strtab
Section header table

The .text section contains machine code generated by a compiler.[38] This part of an ELF file contains the actual instructions to be executed by a computer processor. The .rodata, .data, and .bss sections contain various types of data. The .data and .bss sections respectively contain initialized and uninitialized global variables.[39] The .rodata section contains read-only data, such as format strings for *printf* statements and jump tables for *switch* statements.[40] As will be seen in the subsequent legal analysis, the .text section is particularly relevant for analyzing the GPL-related consequences of static linking. The .rodata, .data, and .bss sections are not particularly relevant from a legal perspective.

The next parts of an ELF file are the .symtab or symbol table and the relocation information, which consists of the .rel.text section and the .rel.data section.[41] A module's symbol table contains information about the symbols defined in that module and symbols referenced by that module.[42] In general, the entries in a module's symbol table will be nonstatic C functions and global variables defined without the static attribute.[43] The symbol table will also contain entries for external variables and other modules referenced by that module.[44] Finally, the symbol table contains entries for symbols defined and referenced exclusively within that particular module.[45] (Examples of these types of entries are functions or variables defined using the static attribute). Each symbol table entry contains information about the applicable symbol, such as the

38. BRYANT & O'HALLARON, *supra* note 19, at 544.
39. *Id.*
40. *Id.*
41. *Id.*
42. *Id.*
43. *Id.* at 545.
44. BRYANT & O'HALLARON, *supra* note 19, at 545.
45. *Id.* at 546.

name, location, size and type (usually either data or function).[46]

In addition to the symbol table, an ELF object file contains relocation information for both the .text (program code) and .data sections.[47] The rel.text section contains a list of locations within the .text section that will need to be modified when the applicable module is linked with other modules to form an executable program.[48] Similarly, the rel.data section contains relocation information for any global variables referenced or defined within that module.[49]

The .debug section and the .line section contain information used to assist programmers when debugging.[50] Specifically, the .debug section contains a debugging symbol table. This symbol table is more extensive than the one contained in .symtab.[51] The debugging symbol table contains additional entries for items such as local variables and typedefs defined in the program.[52] The .line section contains information to map between line numbers in the original C program and machine code instructions in the .text section.[53]

Once the relocatable the object files needed to create an executable file have been generated, the next step is to run the linker with all of those relocatable object files as input. Typically, programs will use generic functions maintained in object code repositories called static libraries.[54] Any static libraries used by a program must also be input to the linker.[55] The linker will resolve all symbol references and associate each reference with exactly one symbol definition in the symbol tables for the input relocatable object files and static libraries.[56] Symbol resolution can be difficult, particularly when there are multiple conflicting defini-

46. *Id.* at 546-47. Symbol name storage is somewhat complex. *Id.* The symbol table entry for a symbol name is a byte offset into another table called the string table. *Id.* An offset in a symbol table entry points to the location of the corresponding name in the string table. *Id.* This additional level of complexity does not have any legal significance. *Id*. The key legal point is that the names of externally defined symbols are located somewhere in an object file. *Id.*

47. BRYANT & O'HALLARON, *supra* note 19, at 545.

48. *Id.*

49. *Id.*

50. *Id.*

51. *Id.*

52. *Id.*

53. BRYANT & O'HALLARON, *supra* note 19, at 545.

54. *Id.* at 553. In practice, all compilation/linking systems provide a way to package related object files into a single file called a static library. *Id.* A typical example of such a library system is the ANSI C libc.a library, which contains numerous functions for performing operations such as standard input/output, string manipulation, and integer math functions. *Id*.

55. BRYANT & O'HALLARON, *supra* note 19, at 555-56.

56. *Id.* at 556.

tions in the input object files and static libraries.[57] Fortunately, a detailed understanding of how these conflicts are resolved is not needed to determine whether the resulting executable file is a derivative work of the linked object files and static libraries.[58]

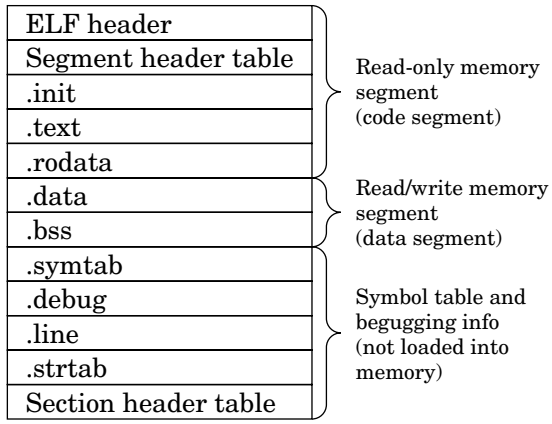
During the symbol resolution phase a linker will scan all of the input object files and static libraries trying to resolve the symbol references that need to be relocated.[59] If a reference to a symbol is found in the symbol table of a particular object file, then that object file is added to the list of object files needed to form the executable.[60] The addition of an object file to this list will mean that the symbols in that object file will also need to be resolved.[61] Resolution of these symbols may in turn require the addition of other object files.[62] This process will iterate until all symbols have been resolved and all object files required to create the executable have been identified.[63] From a legal perspective the important point is that these object files will be copied into the resulting executable file.

In addition to single object files, static libraries (also called archives) can also be provided as input to linkers.[64] If a static library is input to a linker, the linker will attempt to match unresolved symbols against symbols defined by the members of that library.[65] If a member of the library contains a definition for an undefined symbol, then the object file for that member is added to the list of object files needed to form the executable.[66] Once again, an added object file may contain unresolved symbols that will need to be resolved using other object files in the same or other archives.[67] This process will repeat until all symbols have been resolved.[68] After all symbols have been resolved, the list of object files needed to create the executable will have been identified and an executable can be created.[69] If there are any undefined symbols after all of the input object files and archives have been examined, then an error has occurred and an executable will not be created.[70]

---

57. *Id.*
58. *Id.*
59. *Id.*
60. *Id.*
61. BRYANT & O'HALLARON, *supra* note 19, at 556.
62. *Id.*
63. *Id.*
64. *Id.*
65. *Id.*
66. *Id.*
67. BRYANT & O'HALLARON, *supra* note 19, at 556.
68. *Id.*
69. *Id.*
70. *Id.*

Once all symbols have been resolved and a complete list of required object files has been determined, the linker will merge all like sections from all of these object files.[71] Thus, all of the code sections (.text) and data sections (.rodata and .data) are merged into single code and data sections. As described above, for code from archive files, only those object files needed to resolve unresolved symbol references are copied into the executable file. Once all of the sections have been merged, the linker can determine the starting address of each section and compute the final absolute addresses of the symbols in the executable.[72] This is done by processing the code and data relocation information (the .rel.text and .rel.data) information, which specifies the locations in the code and data sections that need to be updated.[73] Once this relocation has been done, the executable object file is in a form that can be copied from disk into computer memory for execution. For an ELF file, the format of a statically linked executable is as follows:[74]



71. *Id.* at 557.

72. BRYANT & O'HALLARON, *supra* note 19, at 557. On Linux systems, the code segment always starts at address 0x08048000. *Id*. The data segment starts at the next 4-KB aligned address after the code segment. *Id*. Since the size of the code segment and all of the data segments are known, and since the starting address of the code and data segments are known, the absolute and relative addresses of all symbols in the executable can be determined. *Id*.

73. *Id.* at 558. ELF files have 11 relocation types that describe the various addressing modes the linker needs to handle. *Id*. The most common relocation types are R_386_PC32 (relocate a reference using a 32-bit PC-relative address) and R_386_32 (relocate a reference using a 32-bit absolute address). *Id*. A PC relative address is an addressing mode in which the target address is an offset from the current run-time value of the program counter (PC). *Id*. For this type of addressing mode, the target address is calculated by adding a 32-bit value to the current value of the PC (which always contains the address of the next instruction in memory). *Id*. An absolute address refers to a specific location in memory. *Id*. Hence the target address for an absolute address is a 32-bit value encoded in the instruction. *Id*.

74. BRYANT & O'HALLARON, *supra* note 19, at 563.

The first thing to notice about a statically linked ELF executable is that it does not contain any relocation information. Also, as compared to a relocatable object file, there is a new section called .init that contains a small function used to call program initialization code.[75] ELF executables are also organized to be easy to load into memory with all of the parts that are actually loaded into memory contained in contiguous segments.[76] Finally, while the symbol table is not loaded into memory for execution, it is still a part of the executable file.[77] As will be recalled, the symbol table contains (together with the .strtab segment) all of the symbols used by a program, including symbols that are imported from external files.

An executable object code file can be run on a UNIX system by typing its name in a shell command line.[78] The shell[79] will recognize that the executable name is not a built-in shell command and will attempt to run the executable by invoking the operating system loader.[80] The operating system loader copies the code and data from the executable object file into memory and then runs the program by jumping to the entry point specified in the ELF Header.[81] The startup code for all C programs is the same and is contained in the *crt1.o* file.[82] The *crt1.o* file contains code that calls the initialization routines in the .text and .init sections.[83] The startup code in the *crtl.o* file also calls the *atexit* function, which appends a list of routines to be called when the executable calls the *exit* function.[84] Once this has all been done, the startup code calls the executable's main routine and execution commences in the actual executable file.[85] After the file finishes normal execution, it will call the *exit* routine, which runs the list of routines created by the *atexit* function and then returns control to the operating system.[86]

From a technical perspective, the creation of a statically linked file is relatively complex. However, as will be discussed later, the legal analy-

75. *Id.*

76. *Id.*

77. *Id.*

78. *Id.* at 564. There are also other ways to invoke a program from another program, such as, for example, using the "exec" function.

79. In UNIX and other similar operating systems, the term "shell" is used to describe an interface between a user and the operating system. The term is typically associated with a text-only user interface. The primary function of a shell is to read and execute commands typed into a terminal.

80. BRYANT & O'HALLARON, *supra* note 19, at 564.

81. *Id.*

82. *Id.*

83. *Id.*

84. *Id.* at 564-65.

85. *Id.* at 565.

86. BRYANT & O'HALLARON, *supra* note 19, at 565.

sis for determining whether a statically linked executable is a derivative work of a GPL-licensed library to which it has been linked is relatively straightforward. For completeness sake, each of the various types of files used to create a statically linked executable should also be considered in the derivative works analysis. These files consist of the source file, the ASCII intermediate file, the assembly language file, the relocatable object file, and the final executable object file.

The source code file will generally have a number of *include* statements in the form "#include <filename>" where "<filename>" is the name of a GPL-licensed library to be linked and used by the calling program. *Include* statements are used to allow a calling program to locate symbols from external programs, modules or libraries. Typically, an *include* statement will reference a "header" file containing definitions for data structures, procedure and function calls and global variables made externally available by a program, module, or library. In addition to *include* statements, a source code file may also contain actual uses of the symbolic references to various procedures, functions, data structures and other objects exported or made externally available by a GPL-licensed program, module or library.

References to symbols exported by a GPL-licensed program will, in the case of symbols for procedures and functions, allow a calling program to make actual use of the capabilities provided by the code associated with those symbols. A program calling a GPL-licensed library may also make use of data structure definitions that specify the way certain data access operations are performed or the way certain memory locations are used. The use of external symbolic references by calling programs will vary from case to case. Therefore, a determination of the legal significance of such uses will require a fact-dependent analysis in each case.

An intermediate ASCII file is created by running the C preprocessor. If run separately, the C preprocessor will generate a file with an ".i" extension. Normally, when a compiler driver is invoked, the intermediate ASCII file is not readily apparent to a software developer since this format is only an intermediate stage in the creation of an object file. (It is normally only created in and utilized from a temporary directory). As discussed above, source code files normally contain a number of *include* statements. An *include* statement is a directive to the C preprocessor to cause the *include* statement to be replaced with the entire contents of the file name referred to in that statement. When linking to a GPL-licensed program, the C preprocessor will cause the contents of header or definitions files associated with that GPL-licensed program to be copied into the .i file of the linking program. The resulting .i file is the same as the source code file for the program except it includes all the definition files referenced using *include* statements.

The next step in the static linking process is compilation.[87] This step is complex and consists of multiple stages. For a typical compiler, the main stages are: lexical analysis; syntax analysis; semantic analysis; intermediate code generation; code optimization; and code generation.[88] The lexical analysis phase separates characters of the source language into logical groups called tokens.[89] These tokens are passed to the next phase, which is the syntax analyzer or parser. The syntax analyzer uses tokens to create an intermediate representation (typically a syntax tree) depicting the grammatical structure of the token stream.[90] The semantic analyzer checks the syntax tree to confirm that the program being compiled has been written in accordance with the rules of the source language.[91] After this is done, the intermediate code generator takes the syntax tree and generates a low-level or machine-like intermediate representation.[92] This intermediate representation is then analyzed for opportunities to improve code efficiency.[93] The final phase of compilation is assembly code generation.[94] In addition to these operations, the compiler is also responsible for various table management or bookkeeping functions.[95] The most important of these functions is the creation of the symbol table.[96] When a compiler is invoked, it typically generates an ASCII assembly language file. The code in an assembly language file may embody non-literal elements copied from linked GPL-licensed programs, modules or libraries.

During the next phase, the compiler driver invokes the assembler. The assembler translates the ASCII assembly language file into a relocatable object file. One of the primary functions during this phase is the translation of the assembly language code into machine code.[97] The assembler will also include data in the relocatable object file that specifies where the object code needs to be modified to reflect the final location of a particular symbol. The linker will subsequently use this information to

87. ALFRED V. AHO, MONICA S. LAM, RAVI SETHI & JEFFREY D. ULLMAN, COMPILERS: PRINCIPALS, TECHNIQUES, & TOOLS 1 (2006) (stating "a compiler is a program that can read a program in one language - the *source* language - and translate it into an equivalent program in another language - the *target* language").

88. *Id.* at 5-11.

89. *Id.* at 6.

90. *Id.* at 8.

91. *Id.*

92. *Id.* at 9.

93. AHO, LAM, SETHI & ULLMAN, *supra* note 87, at 10.

94. *Id.*

95. *Id.* at 11.

96. *Id.*

97. After this translation, the code will consist of 0's and 1's that can be read directly by a computer, as opposed to the mnemonic symbols of a particular machine's assembly language, which cannot be read directly by a computer.

"fix up" these locations once actual symbol addresses are known. As was the case with earlier intermediate files, this file will contain the symbol names being used from any linked GPL-licensed programs, modules or libraries. This file may also embody non-literal portions of any linked GPL-licensed programs, modules or libraries.

The final phase of compilation involves the invocation of the linker. This step causes the creation of an executable object file. Once again, the executable object file will contain the various symbols referenced from any linked GPL-licensed programs, modules or libraries. The code from the calling program may also contain non-literal elements from any linked GPL-licensed programs, modules or libraries. The linker will also copy all of the relocation information from the linked files into the executable file. Finally, and most importantly, the executable object file will contain executable code from the GPL-licensed programs, modules and libraries it uses.[98] This is the *sine qua non* for static linking.

C. Dynamic Linking

Dynamic linking is another method for combining modules. Dynamic linking was developed to address some weaknesses of static linking. Most of these weaknesses are trade-offs for other useful characteristics of static linking. For example, static linking ensures that an executable is self-contained and does not require a particular set of libraries to be present on the machine on which the executable is being invoked.[99] This means the executable invoked is the one actually tested by the developer. Accordingly, when static linking is used, the environment in which an executable is invoked is less likely to affect its behavior.[100]

Despite these benefits, static linking also suffers from a number of drawbacks. For example, the self-contained nature of statically linked programs means it can be difficult to update libraries on which those programs depend.[101] However, the most significant drawback of statically linked files is memory usage. A statically linked file contains all of the code of all of the library routines it utilizes. This means a statically linked executable will require more disk space when stored, more mem-

98. Code from a library or module is only included in an executable object file for a program if that program actually calls routines from that library or module. However, it is reasonable to assume this will happen for most non-trivial uses of a library or module.

99. Christian Collberg, John H. Hartman, Sridivya Babu & Sharath K. Udupa, *SLINKY: Static Linking Reloaded*, 2005 USENIX annual Technical Conference 1 (2005), http://www.cs.utah.edu/classes/csl-sem/old/f05/papers/slinky-usenix05.pdf.

100. *Id.*

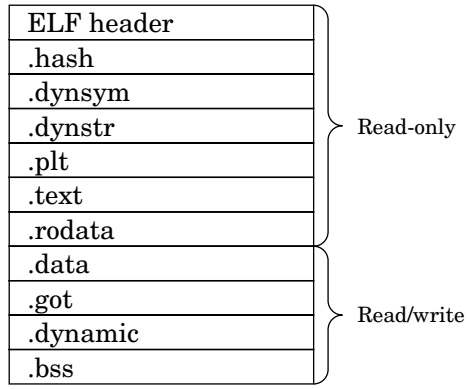
101. Bryant & O'Hallaron, *supra* note 19, at 566. When a library is updated, programs statically linked to an earlier version of that library will need to be re-linked to use the updated version of the library.

ory space when executed, and more network bandwidth when transmitted.[102] For example, almost every C language program uses standard I/O functions such as *printf* and *scanf*.[103] At run-time, the code for these functions will be copied into the .text sections of all of these running processes.[104] On a typical system there can be as many as 50 to a 100 processes running at any given time.[105] This replication is a significant waste of system memory.[106] Dynamic linking addresses these problems by deferring much of the linking process until an executable is actually loaded, and sometime even later than that.[107]

Dynamic linking is based on the concept of shared libraries. A shared library is an object module that can be loaded at any memory address and then linked with other applications that reference symbols in the library. On UNIX systems, shared libraries are usually referred to as shared objects and have the file type ".so."[108] On Windows systems, this same concept is implemented through dynamically linked libraries ("DLLs").[109] Shared objects save memory in two ways.[110] First, only one copy of a shared object is stored in the file system.[111] Second, only one copy of the .text section of a shared library is copied into main memory for execution.[112] Hence, the use of shared objects saves both disk and main memory space.[113]

Dynamically linked object files are similar to statically linked object files and contain much of the same information, such as code, symbolic names, and relocation information. The following diagram shows the structure of an ELF shared library:[114]

102. Collberg, Hartman, Babu & Udupa, *supra* note 99, at 1.
103. BRYANT & O'HALLARON, *supra* note 19, at 566.
104. *Id.*
105. *Id.*
106. *Id.*
107. LEVINE, *supra* note 33, at 205.
108. BRYANT & O'HALLARON, *supra* note 19, at 566.
109. *Id*.
110. *Id*. at 567.
111. *Id*.
112. *Id*.
113. *Id*.
114. LEVINE, *supra* note 33, at 209.



An ELF dynamically linked program looks much like an ELF shared library. The main difference is that a dynamically linked program has an INTERP section near the beginning of the file so the name of the dynamic linker to be used to load the file can be specified.[115] Additionally, a dynamically linked program does not have a ".got" section, because program files are not relocated at run-time.[116]

The process for creating a dynamically linked program is similar to that for creating a statically linked program. In fact, the same tools are used for static and dynamic linking, but these tools are invoked with different command line parameters. For dynamic linking, the compiler driver is given command line parameters directing the compiler driver to create a shared object file. For static linking, the compiler driver is directed to create object files suitable for inclusion in a static library. Importantly, as part of the procedure for creating a shared object, the compiler driver is directed to generate position independent code.[117] Position independent code is code that can be executed at any arbitrary memory location without having to be modified by a linker.[118] Position independent code is created using addressing modes that do not rely on absolute addresses.[119] For example, calls to procedures within the same module can be made using PC-relative addressing.[120] When calling procedures within the same module, the offset from the location of the call to the location of the called procedure will be known (since it is in the same module). Together with the program counter, all of the information needed to call a procedure within the same module will always be known

115. *Id*.
116. *Id*.
117. BRYANT & O'HALLARON, *supra* note 19, at 567.
118. *Id*. at 570.
119. *Id*. at 570-71. An absolute address is a direct numeric reference to a location in memory.
120. This form of addressing generates the address of a called procedure as an offset from the current value in the processor's program counter.

at code generation time. If absolute addresses were used, then these addresses would have to be fixed up by a linker once the location where the program was going to run was known.

The PC-relative addressing technique described above and other similar techniques can handle all references to local symbols. However, problems arise for calls to external symbols because these references are normally not position independent.[121] These problems can be solved using certain "bookkeeping" techniques and implicit information derived from the way in which object files are organized. Since the data segment of an object file is always loaded into memory immediately after the code segment, the distance between any location in the code segment and any location in the data segment is a run-time constant – meaning this distance will be independent of the location where an object file is loaded in memory.[122] A compiler takes advantage of this implicit knowledge by placing two special data structures in the object file.[123] These data structures are used by the compiler to create position independent code for calling external procedures and referencing external global variables.[124]

The first of these data structures is called the global offset table or GOT.[125] The GOT contains one entry for each global data object referenced by an object module.[126] The compiler also generates a relocation record for each of these entries.[127] At load-time, the dynamic linker will process these relocation records and store the absolute address of each symbol in the corresponding entry for that symbol in the GOT.[128] Once this has been done, global variables can be accessed indirectly through position independent code that uses the GOT.[129] This technique works because the location and size of the GOT are known at compile time. Therefore, position independent code can be written that references the GOT entry for a particular symbol. This type of indirect addressing works because the absolute address for a symbol can be determined and stored in the GOT entry for that symbol before execution occurs. At load-time the absolute addresses for these positions will be known and can be written into the GOT so that at run-time the proper absolute address can be used by the position independent executable code in the .text

121. BRYANT & O'HALLARON, *supra* note 19, at 571.
122. *Id.* at 572.
123. *Id.*
124. *Id.*
125. *Id.*
126. *Id.*
127. BRYANT & O'HALLARON, *supra* note 19, at 572.
128. *Id.*
129. *Id.*

section.[130]

A similar, though somewhat more complex technique, is used when calling external procedures. An additional data structure called the procedure linkage table or PLT is used in the binding process for external procedure calls.[131] Unlike the GOT, the PLT is contained in the .text segment and the PLT is itself position independent code.[132] Since the location of the PLT within the .text section is known at compile time, each reference to an external procedure can be made using position independent code that refers to that procedure's entry in the PLT. Unlike the GOT where all of the entries are bound at load-time, the dynamic linker defers binding of procedure addresses until the first time each procedure is called. This is called lazy binding.[133] The first time a program calls a procedure, the PLT entry for that routine causes the invocation of the dynamic linker.[134] The dynamic linker will resolve the address of the procedure that caused the jump to the linker and will store that address in the GOT entry for that procedure.[135] While the first invocation of each procedure is quite slow due to the call to the dynamic linker and the resulting address resolution, subsequent calls to each resolved procedure will be much quicker and involve only a single extra jump to the PLT.[136]

The way in which the dynamic linker is invoked and the way in which the absolute address of a particular procedure is resolved is quite complicated. Before explaining how this is done, a further description of the GOT is required. The first three entries of the GOT contain special information.[137] The first entry in the GOT contains the address of the .dynamic segment, which contains information the dynamic linker uses to bind procedure addresses.[138] The second entry in the GOT contains information identifying the module being dynamically linked.[139] The third entry in the GOT contains an entry point into the lazy binding code of the dynamic linker.[140] As described above, each external procedure called by a module will have an entry in the GOT. Each of these procedures will also have an entry in the PLT.[141] The entries in the PLT are initialized with a special sequence of instructions that will cause the procedure associated with that PLT entry to be dynamically resolved. The

130. *Id.*
131. *Id.* at 573.
132. LEVINE, *supra* note 33, at 213.
133. BRYANT & O'HALLARON, *supra* note 19, at 573.
134. *Id.* at 573-74.
135. *Id.* at 574.
136. *Id.*
137. *Id.* at 573.
138. *Id.*
139. BRYANT & O'HALLARON, *supra* note 19, at 573.
140. *Id.*
141. LEVINE, *supra* note 33, at 213.

code in the .text segment that calls an external procedure will point to the first instruction in the PLT entry for that external procedure.[142]

The addresses for external procedures in a program are only resolved at run-time. The first call to an external procedure will cause a jump to the location of that procedure's entry in the PLT and the code at that location will be executed. The first instruction in each unresolved PLT entry is a jump to the address contained in the corresponding entry for that procedure in the GOT.[143] In effect, this first jump does nothing because the GOT entry is initialized to point to the next instruction in the PLT entry.[144] This next instruction is a "push" instruction that pushes (or stores) a value on the run-time stack.[145] The value stored on the run-time stack is an offset value that indirectly identifies the symbol to be resolved and the GOT entry into which the resolved address should be loaded.[146] Once this instruction is executed, the next instruction jumps to PLT[0].[147] The instructions at PLT[0] push another code on the stack.[148] This code identifies the program calling the dynamic linker that, in this case, is the program asking for dynamic address resolution. The program then jumps into the dynamic linker with the two identifying codes that just have been loaded at the top of the stack.[149] The return address of the procedure that made the call is also pushed onto the stack.[150]

The first action performed by the dynamic linker is to save all the registers being used by the calling program.[151] The two identifying words stored on the stack are used to locate the procedure whose address is being resolved.[152] Once the dynamic linker has obtained that address, it stores the address in the entry for that procedure in the GOT of the calling program.[153] After has been done, the dynamic linker restores all of the saved registers and pops the two words of identifying information

142. *Id.*

143. *Id.* at 213-14. This is true for each entry in the PLT except for PLT[0], PLT[1], and PLT[2], which are initialized with special instructions that call the dynamic linker and identify the program requesting symbol resolution. *Id.* at 214.

144. LEVINE, *supra* note 33, at 214.

145. *Id.* at 213-14. The "stack" is a special temporary memory location used for the execution of a program.

146. LEVINE, *supra* note 33, at 214.

147. *Id*.

148. *Id*.

149. *Id*.

150. *Id*.

151. *Id*. The registers are saved so the current state of the calling routine can be restored once the dynamic linker has performed its functions. If the registers are not saved, the calling program will not be able to operate properly once the dynamic linker returns control.

152. LEVINE, *supra* note 33, at 214.

153. *Id*. at 214-15.

off the top of the stack.[154] The dynamic linker then jumps to the address of the routine that was just resolved and continues execution. The next time this procedure is called, the GOT entry will contain the actual address of the procedure. Instead of jumping back to the PLT and calling the dynamic linker again, the GOT entry will be used to jump to the called routine.[155] Using the techniques just described, it is possible to generate position independent code that can run at arbitrary locations within memory. The use of position independent code allows the creation of shared libraries that, in turn, allows dynamic linking of programs.

The final step in the dynamic linking process involves the actual loading of a dynamically linkable program into memory. When an operating system runs a dynamically linkable program it first maps the program's pages into memory.[156] The operating system will detect that the program has an INTERPRETER section.[157] This section indicates to the operating system that it needs to invoke an interpreter program and also identifies the interpreter to be used.[158] Typically, this interpreter will be *ld.so*, which, as indicated by the file extension, is also a shared object.[159] When *ld.so* is invoked, certain information is passed to it, such as the entry point for the program being loaded. The first thing *ld.so* does when it is invoked is to use certain bootstrap code to relocate its own code and data references.[160] Once this is done *ld.so* can load the program to be executed and all the libraries needed by that program. The dynamic linker starts this process by initializing a chain of symbol tables with a pointer to the symbol table of the program being linked and a pointer to the symbol table of the linker itself.[161] The linker then identifies all of the libraries needed by the program being loaded. A pointer in the linked program header allows the linker to locate this information.[162] Information about any linked libraries is contained in two structures. One is a list of DT_NEEDED entries; each of these entries contains an offset in the DT_STRTAB data structure, which points to the start of the name of the required library.[163] The DT_STRTAB table contains charac-

154. *Id*. at 215.
155. *Id*.
156. *Id*. at 210.
157. *Id*. at 210.
158. LEVINE, *supra* note 33, at 210.
159. *Id*.
160. *Id*.
161. *Id*.
162. *Id*.
163. *Id*. at 210. The DT_STRTAB structure contains actual character string names and is similar to the STRTAB structure used in a statically linked file. Unlike the STRTAB structure, which contains symbol names, the DT_STRTAB structure contains linked library names.

ter strings that correspond to the names of required libraries.

Once the dynamic linker has identified the name of a library, it searches for that library in the file system. This process is more complicated than it seems; fortunately, a detailed understanding of how this search is performed is not required for a derivative works analysis. After locating a library in the file system, the dynamic linker opens the file for that library and loads the appropriate parts into memory.[164] At a high level, the loading of a library consists of allocating memory space for the library's text and data segments and then copying those segments into memory.[165] The linker will also allocate space for the .bss section and initialize it to all zeros.[166] The library's symbol table will be also added to the chain of symbol tables the linker is building for the program being loaded. The linker also will examine the dynamic segment for the newly loaded library to determine if that library depends on any other libraries.[167] Therefore, the loading of one library can cause a cascade effect whereby a number of other libraries must also be loaded. This process will continue until all of the libraries required by the program being loaded, whether directly or indirectly, have been loaded. Once this is done, the linker will have a complete symbol table.[168] As a practical matter, this complete symbol table will be in the form of a linked list consisting of the *ld.so* symbol table, the program's symbol table, and the symbol tables of all the libraries required by that program.

After assembling the global symbol table, the linker checks the program and each library and processes the relocation entries, populating all GOT entries and performing any needed relocations in the various data segments.[169] Once the data relocations have been completed, the

164. LEVINE, *supra* note 33, at 212. If a library has already been loaded into memory, there is no need to read it in from the file system and allocate new memory for it. Instead, the version of the library already been loaded into memory is used.

165. LEVINE, *supra* note 33, at 212.

166. *Id.* As described earlier, the .bss section does not occupy any space in an object file. It is simply a place holder. However, once a program or library is loaded, memory needs to be allocated for this section. By definition, all .bss locations are initialized to zero. Therefore, part of the process for loading the .bss section is the overwriting of the current contents of the allocated memory with zeros. This also explains why the .bss section is not allocated any space in an object file. Since the .bss is always initialized to zeros, there is no need to store this information in the object file because it can be easily re-created at load-time. Since the size of the .bss section is recorded in the object file, the .bss section can always be fully re-created by simply allocating the appropriate amount of memory and zeroing it out. This differs from the .text section and other data sections since they have variable content that must be copied into memory.

167. LEVINE, *supra* note 33, at 212.

168. *Id*.

169. *Id*.

linker will run the program and library initialization code.[170] At this point no address resolution has been done for any procedure calls. Instead, the lazy binding approach is taken using the GOT and PLT as described earlier. The reason for taking this approach is that programs and libraries tend to contain large numbers of functions. During any particular invocation of a program, many of these functions will never be called; hence, the effort to resolve these unused procedures would be wasted.[171]

Having concluded the technical description of dynamic linking, it is now possible to identify the characteristics of a dynamically linked file that will be important in a derivative works analysis. As will be discussed later, and unlike static linking, determining whether a dynamically linked program is a derivative work of the libraries it calls is not straightforward. While there are a number of similar steps and file types in static and dynamic linking, there are key differences that affect the legal analysis.

As with the static linking analysis, the derivative works analysis for dynamic linking will assume that a non-GPL licensed program is linking to a GPL-licensed library. The source code files for both dynamically and statically linked programs will generally have *include* statements in the form "#include <filename>" so that the various external symbols used by those programs can be located in the appropriate libraries. As with statically linked programs, the source code of a dynamically linked program will also typically contain symbolic references to various procedures, functions, data structures and other objects exported or made externally available by the libraries being used by the program.

Dynamic linking also requires the creation of an intermediate ASCII file generated using the C pre-processor. The running of the C pre-processor will cause the contents of files referenced in *include* statements (including any files that are in turn referenced in *include* statements in any of those files) to be copied into the intermediate ASCII file. As with static linking, this file will be essentially the same as the source code file, except it will include any files that have been referenced directly or indirectly by *include* statements in the original source code file.

The next step in the process is the compilation phase. Once again, an ASCII assembly language file is created. Indirectly, the code in this assembly language file may embody the various data structures and definitions used by the calling program and defined in any referenced libraries. This file will also contain the various symbols utilized within the

170. *Id*. This is the code located in the .init section, which was discussed in connection with the description of the ELF dynamic object code file structure.

171. LEVINE, *supra* note 33, at 213.

calling program. These symbols will be stored in the dynamic symbol table.

During the next phase of the compilation and dynamic linking process, the compiler driver invokes the assembler. The assembler translates the ASCII assembly language file into a program object file. As in the case of static linking, the assembler will also include data in the relocatable object file that indicates those locations where the object code needs to be modified to reflect the final location of a particular symbol. However, the format of this information is different from that in static linking. Unlike dynamic linking, static linking does not use either a GOT or a PLT. Similar to static linking, this file will contain the names of symbols referenced from called libraries. This file will also contain information in the dynamic section that describes the various libraries on which the program depends. As with statically linked machine code, the machine code in this file may embody non-literal elements of the called libraries.

The final phase involves the invocation of the linker. In dynamic linking, an executable image is created in memory rather than in an object file. When dynamic linking is done, the program created at development-time is incomplete in the sense that there is still a significant amount of linkage to be done at load-time or sometimes even at run-time to allow the program to run correctly. This contrasts with static linking where the object file created at development-time is complete and can be executed without any further linkage. A dynamically linked object code program will have a dynamic symbol table containing the various symbols called from any GPL-licensed libraries it uses. The machine code in this program may also embody non-literal elements from any GPL-licensed libraries. A dynamically linked program will, however, only contain relocation information for itself. Any relocation information for any shared libraries on which the program depends will be contained in those libraries. Similarly, the GOT and PLT do not contain any information from the shared objects on which the program depends. Finally, and most importantly, a dynamically linked executable object file will not contain any code from any shared objects to which it is linking. This code remains with the applicable shared objects and only one common copy is shared among all of the dynamically linked programs referencing those shared objects. This is the *sine qua non* for dynamic linking.

For completeness, the executable image created by a dynamic linker also should be considered. The executable image is an amalgam of code, data and register settings in computer memory. It is important to remember that the executable image does not correspond to the object file distributed by a computer software vendor. The object file distributed by a computer software vendor is an ELF file as described earlier in this section. The image of a running program in memory is significantly dif-

ferent. That image consists of the program itself as well as all of the shared objects it depends on that may or may not have been provided by the developer of the program.[172] Therefore, an executing program is really a patchwork of code and data segments in a running computer. When a program is loaded, many of the shared objects it depends on may already be in memory. When this happens, the dynamic linker simply performs the operations needed to allocate the various private data segments, link the required symbol tables, and perform the necessary relocations. If a program being loaded requires a shared object that is not already loaded in memory, then that shared object will be loaded. It is not correct to say that the shared object will be copied into the program being loaded, since, once loaded, the shared object will have its own memory location and will be available for use by any other programs on that computer. In reality, a shared object is loaded on its own, and the program that caused it to be loaded is modified to reference that newly loaded shared object. The linking program and the shared object to which it is linking always occupy separate memory locations but within the same memory space.

As evidenced by the foregoing description, the object code of a linked library is not copied into a dynamically linking program – even when that dynamically linking program is running on a computer. However, the dynamically linking program may contain various symbol names defined in the libraries to which it links. Accordingly, in determining whether a dynamically linking program is a derivative work of the libraries to which it links, it is important to understand how copyright law protects methods of operation (the mechanisms that allow a shared object to be used by another program), data structures (for non-literal elements that may be embodied in a calling program), and symbolic names (such as those used for procedure calls and variable and field names).

D. Inter-process Communication

The final type of program-to-program interaction to be examined is network or inter-process communication. Typically, networked programs use client-server architectures.[173] A client-server system describes a relationship between two types of computer programs in which one program, the client, is designed to submit service requests to another program, the server.[174] The server, in turn, is designed to receive and

172. One can also argue that the image includes the operating system since significant operating system functionality is required to run a dynamically linked program.

173. Bryant & O'Hallaron, *supra* note 19, at 802. Other architectures are possible – such as, for example, peer-to-peer architectures.

174. *Id*.

fulfill service requests.[175] The format and mechanics for the exchange of service requests and replies between a client and a server are specified in a protocol.[176] There are many well-known examples of client-server architectures and corresponding protocols that will be familiar to everyday computer users. For example, web browsers and web servers interact using the Hypertext Transfer Protocol, commonly known as HTTP.[177] Computer users can exchange files using the File Transfer Protocol ("FTP"). Numerous other protocols exist for capabilities such as remote procedure call ("RPC"), domain name resolution ("DNS"), and various features of electronic mail (SMTP and MIME, for example).[178] The most significant protocols for program-to-program communication are based on an underlying communication protocol suite called Transmission Control Protocol/Internet Protocol or TCP/IP. These are the underlying communications protocols for the global Internet.[179] The definition of a protocol is usually set forth in a document called a request for comments or RFC. RFCs began in 1969 as part of the original ARPANET project.[180] Since that time, RFCs have become the official channel for standards and other publications of the Internet Engineering Task Force ("IETF").[181] The IETF is a large international community of network designers, operators, vendors and researchers who are involved in the operation and evolution of the Internet.[182] Technical work within the IETF is done through working groups.[183] For example, there are working groups focused on issues such as routing, transport and security.[184]

Many major computer operating systems, including all UNIX variants, Macintosh and Windows, make network communications capabilities available to applications using an API called *sockets*.[185] Researchers at the University of California, Berkley originally developed the *sockets*

175. *Id*.

176. W. RICHARD STEVENS, BILL FENNER & ANDREW M. RUDOFF, UNIX NETWORK PROGRAMMING: THE SOCKETS NETWORKING API 3 (2004).

177. BRYANT & O'HALLARON, *supra* note 19, at 826.

178. *See* RFC Editor, Official Internet Protocol Standards, http://rfc-editor.org/rfcxx00.html (last visited May 4, 2010) (providing a comprehensive list of RFCs).

179. BRYANT & O'HALLARON, *supra* note 19, at 807-08.

180. Wikipedia, *Requests for Comments: History,* http://en.wikipedia.org/wiki/Requests_for_comments#History (last visited Apr. 26, 2009).

181. *Id*.

182. IETF, *Overview of the IETF*, http://www.ietf.org/overview.html (last visited Apr. 26, 2009).

183. *Id*.

184. IETF, *Active IETF Working Groups*, http://datatracker.ietf.org/wg/ (last visited Apr. 26, 2009).

185. BRYANT & O'HALLARON, *supra* note 19, at 816; STEVENS, FENNER & RUDOFF, *supra* note 176, at 20; Brian "Beej" Hall, *Using Internet Sockets,* BEEJ'S GUIDE TO NETWORK PROGRAMMING 5 (Sept. 8 2009), http://beej.us/guide/bgnet/output/print/bgnet_USLetter_2.pdf.

API in the early 1980s.[186] While the *sockets* API was designed to work with any underlying protocol, it was first implemented using TCP/IP.[187] Today, TCP/IP remains the most significant transport and network layer protocol combination.

Before examining how the *sockets* API is used in a client-server system, some basic concepts about network communication need to be explained. A network consists of hardware and software components that work together to allow communications between physically separate computers.[188] There are many different kinds of computer networks and these different types of networks have varying geographic coverage. For example, Local Area Networks, or LANs, can be used for communications within a room, building or campus.[189] For communications on a city-sized scale, a Metropolitan Area Network, or MAN, would be more appropriate.[190] Finally, for country, continent or planet-wide communication, Wide Area Networks, or WANs, are used.[191] The way in which computers are connected can be varied to suit size and throughput requirements. Typical media used for network connectivity are copper wire, fiber optics, wireless, microwave, and satellite communication.[192] Despite the large number of different types of computer networks and the varying geographic scales and implementations, the semantics for setting up *sockets*-based communication between two programs is relatively straightforward and well shielded from the details of the underlying physical network.

The first major task for a designer of a message-based application is to identify a particular instance of a specific program in a network to which a message is to be sent. In a TCP/IP-based network this is done using an IP address. An IP address is an unsigned-32 bit integer used to identify a host or computer.[193] Sometimes an IP address is displayed for human viewing in dotted decimal notation, where each byte is represented by its decimal value and separated from the other bytes by peri-

186. BRYANT & O'HALLARON, *supra* note 19, at 816.

187. *See* BRYANT & O'HALLARON, *supra* note 19, at 816; *See also*, STEVENS, FENNER & RUDOFF, *supra* note 176, at 20.

188. *See* BRYANT & O'HALLARON, *supra* note 19, at 803.

189. ANDREW S. TANENBAUM, COMPUTER NETWORKS 16-17 (4th ed. 2003). A typical example of a LAN is the IEEE 802.3 or Ethernet.

190. TANENBAUM, *supra* note 189, at 18. The best-known example of a MAN is the cable television network found in most cities.

191. TANENBAUM, *supra* note 189, at 19-21. Wide area networks tend to be much more heterogeneous, with various switching elements and transmission lines connected in irregular topologies. The Internet and the Public Switched Telephone Network are examples of well-known WANs.

192. *See generally*, TANENBAUM, *supra* note 189, at ch. 2.

193. *See* BRYANT & O'HALLARON, *supra* note 19, at 808-09.

ods.[194] More commonly, however, IP addresses are publicly presented to humans as domain names. Domain names are sequences of characters intended to provide a friendly and more easily remembered way of denoting a particular IP address.[195] Examples of well-known domain names are google.com, amazon.com and yahoo.com. A distributed worldwide database known as the Domain Naming System is maintained to store mappings between dotted decimal representations of IP addresses and domain name representations of IP addresses.[196] The Internet Corporation for Assigned Names and Numbers ("ICANN") is currently responsible for managing the Domain Naming System.[197] ICANN is a non-profit organization that assumed the DNS management functions under a contract from the United States Department of Commerce.[198] Previously, part of the United States government performed these management functions.[199]

Domain names are formatted in a hierarchy consisting of first-level, second-level and third-level domain names.[200] First-level domain names are a defined set of names that provide a very high-level indication about the owner of a particular domain name. Typical first-level domain names are .gov (a government entity), .edu (an educational institution), and .com (a commercial entity).[201] Second level domain names, such as "amazon" and "google" are, subject to certain restrictions, assigned on a first-come, first-served basis by entities known as domain registrars.[202] Domain registrars are appointed and operate under the auspices of ICANN or a national top-level domain authority or both.[203] Using information maintained in the Domain Naming System, the IP address of a particular host computer can be obtained by using its domain name, or conversely, the domain name of a particular host computer can be obtained by using its dotted decimal IP address.[204] These capabilities are important because an Internet communication connection consists of a pair of endpoints specified by their IP addresses and a communication

194. *Id.* at 809. For example, the IP address 192.0.34.163 is used by the Internet Corporation for Assigned Names and Numbers (ICANN) at http://www.icann.org.

195. BRYANT & O'HALLARON, *supra* note 19, at 811.

196. *Id.* at 812.

197. Internet Corporation for Assigned Names and Numbers, ICAAN About, http://www.icann.org/tr/english.html (last visited Apr. 26, 2009).

198. Wikipedia, ICANN, http://en.wikipedia.org/wiki/Icann (last visited Apr. 26, 2009).

199. Caslon Analytics, ICANN and the UDRP, http://www.caslon.com.au/icannprofile1.htm (last visited July 19, 2010).

200. BRYANT & O'HALLARON, *supra* note 19, at 811.

201. *Id.*

202. *Id.* at 812.

203. Wikipedia, Domain Name Registrar, http://en.wikipedia.org/wiki/Domain_name_registrar (last visited Apr. 26, 2009).

204. BRYANT & O'HALLARON, *supra* note 19, at 812.

stream passing between the endpoints using a chosen protocol.[205]

The basic data structure used for communication in the *sockets* system is a *socket*. A *socket* is an endpoint of an Internet connection. A *socket* address consists of an IP address, as described above, and an additional sixteen-bit addressing unit called a *port*.[206] *Port* numbers are used because at any given time on a particular host computer there may be multiple processes using a particular transport protocol. A *port* is used to differentiate between these processes.[207] Servers for well-known client-server protocols, such as FTP and HTTP, are assigned well-known and fixed *port* numbers.[208] Clients, on the other hand, are usually assigned short-lived and varying *port* numbers, known as ephemeral *ports*.[209]

Internet clients and servers communicate by sending and receiving streams of data between connections. A connection is uniquely identified by the *socket* addresses of its two end points.[210] The IP address portion of each *socket* will identify the host computer on which the communicating process is a resident.[211] The *port* number will identify the particular process on a host computer to which a communication stream is to be directed.[212]

A connection is created between two processes using a number of function calls available in the *sockets* API.[213] The first step in creating a connection is to create a *socket* at each end of the connection.[214] Internet *socket* addresses are stored in 16-byte structures of the type *sock_addr_in*.[215] The following is the definition of the *sockaddr_in* data structure:

```
struct sockaddr_in{
unsigned short      sin_family;   /* address family (always AF_INET)   */
unsigned short      sin_port;     /* port number                       */
struct in_addr      sin_addr;     /* IP address                        */
unsigned char       sin_zero[8]   /* pad to sizeof (struct sockaddr)   */
                                                                        };
```
[216]

205. *Id.* at 815.

206. *Id.*

207. *See* STEVENS, FENNER & RUDOFF, *supra* note 176, at 20.

208. *Id.* For example, the FTP service is assigned the well-known *port* number 21 (decimal), while the HTTP service is assigned the well-known *port* number 80 (decimal). *Id.*

209. *Id.* The value of a *port* assigned to a client is normally not significant. *Id.* The important point is that the *port* number is unique so it can identify a particular client to whom a communication stream is to be directed. *Id.*

210. BRYANT & O'HALLARON, *supra* note 19, at 815.

211. *Id.*

212. *Id.*

213. *Id.* at 817-23.

214. *Id.* at 818.

215. *Id.* at 817.

216. BRYANT & O'HALLARON, *supra* note 19, at 817.

The key components of this data structure are the IP address (*sin_addr*) and the *port* number (*sin_port*).[217] A *socket* is created by calling the *socket* function that will return a *socket* descriptor that can be then used by a communicating process. Below is the definition of the *socket* function and an example of how it would be typically invoked in the source code of a client program:

int socket (int domain, int type, int protocol);

clientfd = socket (AF_INET, SOCK_STREAM, 0);[218]

The definition (the first line above) indicates to a programmer that the *socket* function takes three parameters. The first parameter specifies the addressing format or domain to be used.[219] The second parameter specifies the type of *socket* to be created. The two most common choices are stream *sockets* and datagram *sockets*.[220] The final parameter specifies the communication protocol the *socket* will use.[221] The invocation of the *socket* function (the second line above) is a command to the operating system to create a TCP-based *socket* and to store the file descriptor for that *socket* in the variable *clientfd*.[222]

The first parameter tells the operating system that the requested *socket* is to be in the Internet domain.[223] The second parameter tells the operating system that the requested *socket* is to be an endpoint for an Internet connection.[224] The third parameter is a default value that tells the operating system to select the protocol that is most appropriate given the values of the first two parameters.[225] In this case, for a stream *socket* in the Internet domain, the most appropriate protocol is TCP. For future activities using this *socket*, the programmer will use the value stored in the variable *clientfd*. The number stored in *clientfd* is a short-

217. *Id.*

218. *Id.* at 818.

219. Samuel J. Leffler, et. al., *An Advanced 4.4 BSD Interprocess Communication Tutorial* 4, http://www.cs.iupui.edu/~cchang/536.01/advanced-ipc.pdf. Two commonly used addressing schemes are AF_INET and AF_UNIX. *Id.* AF_INET is the addressing scheme for the Internet domain (IP addresses and ports) and AF_UNIX is the addressing scheme for the UNIX domain (path names and families). *Id.*

220. *See* Hall, *supra* note 185, at 5. A stream *socket* is a *socket* that is capable of two-way reliable communication. *Id.* In this context, reliable means the data transmitted over this type of *socket* will arrive in the order it was sent and will not be duplicated. *Id.* Reliable also means that if a portion of the data is not received, the sender will be informed of this failure and that portion of the data will be re-transmitted until it is successfully received. *Id.* A datagram *socket* (sometimes referred to as a connectionless *socket*) also supports a bi-directional data flow. *Id.* However, for communication done using a datagram *socket*, the data flow is not guaranteed to be sequenced, reliable or unduplicated. *Id.*

221. BRYANT & O'HALLARON, *supra* note 19, at 818.

222. Leffler, et. al., *supra* note 219, at 4.

223. BRYANT & O'HALLARON, *supra* note 19, at 818.

224. *Id.*

225. *See* Hall, *supra* note 185, at 13.

hand reference that tells the operating system which *socket* is to be used for the activity being requested.

Once a client *socket* has been created, the next operation is to connect that client to a server to allow communications with that server. This is done using the *connect* function, which is defined below:

int connect (int sockfd, struct sockaddr *serv_addr, int addrlen);[226]

The *connect* function will cause the operating system to attempt to establish an Internet connection with the server whose *socket* address is stored in the parameter *serv_addr*.[227] If a successful connection is made to the server, then reading and writing (i.e. receiving and sending) can be done using the *sockfd* descriptor.[228] The Internet address to which a connection is to be made may be hard-coded into the parameter passed to the *connect* function, or it can be determined on the fly by using a function such as *getaddrinfo*.[229] Once a connection has been established, messages can be exchanged using the *send* and *recv* functions. The definitions for these functions are set forth below:

int send (int sockfd, const void *buff, size_t nbytes, int flags);

int recv (int sockfd, void *buff, size_t nbytes, int flags);[230]

For the *recv* function, *sockfd* is the *socket* descriptor of the *socket* from which data is to be read; *buff* is a buffer or storage area where the received data is to be written; and *size_t* is the maximum number of bytes to be read.[231] The final parameter is usually set to 0.[232] The *recv* function returns the number of bytes actually read into the buffer or -1 if an error has occurred.[233] For the *send* function, *sockfd* is the *socket* descriptor of the *socket* to which data is to be sent; *buff* is a pointer to the data that is to be transmitted; and *size_t* is the length of the data to be transmitted measured in bytes.[234] Once again, the *flags* parameter is typically set to 0.[235] *Send* returns the number of bytes sent or -1 in the case of an error.[236] The *send* and *recv* functions may send or receive fewer than the number of bytes specified in the *size_t* parameter.[237] Usually when this happens, the *send* or *recv* function will need to be

226. BRYANT & O'HALLARON, *supra* note 19, at 818.

227. *Id.*

228. *Id.* at 819. From a UNIX programming perspective, a *socket* is treated in much the same way as an open file. *Id.*

229. *See* Hall, *supra* note 185, at 16. The *port* number for the destination server must also be determined. *Id.*

230. STEVENS, FENNER & RUDOFF, *supra* note 176, at 387.

231. Hall, *supra* note 185, at 24.

232. *Id.*

233. *Id.*

234. *Id.* at 23.

235. *Id.*

236. *Id.* at 24.

237. Hall, *supra* note 185, at 24.

called again to either send or receive the remaining data.[238]

The semantics for setting up a connection on the server side of a client-server pair are different.[239] In the server case, the operations required to set up a connection are *bind*, *listen,* and *accept*. The following are the definitions for these functions:

int bind (int sockfd, struct sockaddr *my_addr, int addrlen);

int listen (int sockfd, int backlog);

int accept (int listenfd, struct sockaddr *addr, int *addrlen);[240]

The *bind* function causes the operating system to associate the *socket* address for a server (*my_addr*) with a *socket* descriptor *sockfd*.[241] The third parameter gives the size of the *sockaddr_in* structure.[242] Server *sockets* are passive in the sense that rather than initiating a connection, a server *socket* passively waits to receive connection requests from clients.[243] By default, the operating system assumes that any calls to the *socket* function are requests for an active or client *socket*.[244] Accordingly, a server needs to inform the operating system that it wants to create a passive *socket* – this is done using the *listen* command.[245] The *listen* command tells the operating system to convert an active *socket* into a listening *socket* capable of accepting client connections.[246] As with other *sockets*-related functions, the *sockfd* parameter identifies the *socket* to be acted upon.[247] The backlog parameter identifies the number of connection requests that can be queued before the operating system stops accepting further connection requests for that *socket*.[248] Once a *socket* has been bound to a particular address and converted to a listening *socket*, it is ready to wait for connection requests from clients.[249] This is done by calling the *accept* function.[250] The *accept* function waits for client connection requests for the listening *socket* specified by the *listenfd* parameter.[251] When a connection request arrives, the *socket* address information for the client is stored in the *addr* parameter and the function returns a connected descriptor that can be used to communicate with the

238. *Id.*
239. This description assumes that a server *socket* has already been created using a call to the *socket* function.
240. *See* BRYANT & O'HALLARON, *supra* note 19, at 819-821.
241. *Id.* at 820.
242. *Id.*
243. *Id.*
244. *Id.*
245. *Id.*
246. BRYANT & O'HALLARON, *supra* note 19, at 820.
247. *Id.*
248. *Id.* at 820-21.
249. *Id.* at 821.
250. *Id.*
251. *Id.*

client.[252] The server can now communicate with the client by using the connected descriptor and the *send* and *recv* functions.[253]

The *sockets* API allows a programmer to create a client-server pair that can communicate using a transport protocol, such as TCP.[254] The data exchanged by a client and server will be in the form of a byte stream; however, when that byte stream is used in connection with a higher level protocol, it will contain information that has meaning within the context of that higher-level protocol. The higher-level protocol may be one defined in an RFC, such as HTTP, FTP or Simple Mail Transfer Protocol ("SMTP"), or it may be a non-standards-based protocol that has been defined for a specific client-server application.

As was the case with statically and dynamically linked programs, the source code for communicating client and server programs will almost certainly contain *include* statements in the form "#include <filename>".[255] These *include* statements allow client and server programs access to various data structures and procedure definitions for the *sockets* API and other APIs that may be used by those programs. The source code for the client and server programs will also contain actual calls to the procedures made externally available by these APIs. In general, use of the *sockets* API (or any other API or interface defined in other well-known standards such as FTP, HTTP or SMTP) will involve the use of a non-GPL licensed set of definitions and functions. If someone develops a GPL-licensed implementation of the *sockets* API (or any other well-known third-party defined interface) the *sockets* data structure definitions and procedure calls used within that implementation will not be original to that implementation.[256] Since these materials cannot be original to the author of any such GPL-licensed implementation, such an author cannot claim copyright to them.[257]

---

252. BRYANT & O'HALLARON, *supra* note 19, at 821-22. Each time a connection is created between a server and client, the *accept* function will return a new connected descriptor. A connected descriptor is used to communicate with a specific client and exists only as long as it takes to service the client request. By contrast, a listening descriptor is used only to set up connections with clients and once a connection has been established, the listening descriptor is not used to exchange data with those clients. Typically, a listening descriptor is created only once and exists for the lifetime of a particular instance of a server.

253. Hall, *supra* note 185, at 22.

254. BRYANT & O'HALLARON, *supra* note 19, at 816.

255. On a UNIX system, the *include* file reference would be to the sys/socket.h file.

256. If such a GPL-licensed implementation does not copy the definitions and procedures in the specification for the particular API, interface or protocol, it will not work properly. For software that implements a well-known standard, strict compliance with the standard is required if the implementation is going to work properly with other software.

257. The underlying code can be original in the sense that it need not have been copied from another implementation. The main constraint on the code is that it has to provide the

When *socket* calls or higher-level communication APIs are used there is no need for any server source or object code to be incorporated into a client. The client and server operate separately and can even operate on different computers. Instead of sharing object code or exchanging process control, a client and server interact by exchanging messages. The key to this interaction is that the client and server must use the same protocol. When a client communicates with a particular server, the client will need to submit specific messages in a specific sequence. By submitting the proper messages in the expected sequence a client can receive services from a server. These messages and the required sequencing are usually defined in a header file, which can be referenced within a program using an *include* statement. A definitions file can be used to specify a protocol that is unique to a given server or the server can use a definitions file for a well-known or third-party protocol. This latter situation can arise if a programmer is creating a GPL-licensed implementation of a well-known client-server architecture or a GPL-licensed component of a well-known protocol stack.[258]

In the first scenario described above, a client program calling a GPL-licensed server will need to use a protocol that has been specifically designed for that server. The use of the various data structures associated with a unique protocol adopted by such a GPL-licensed server is likely to be indirectly reflected in the object code of the client program. For example, the use of data structures developed specifically for a particular server may cause object code to be generated and memory to be allocated within the client program based on memory mappings specified by those data structures. Accordingly, such a client program may copy non-literal elements of a server program with which it wishes to communicate. Therefore, when determining whether a client program is a derivative work of a server program with which it communicates, it will be important to understand how copyright law protects data structures and protocols.

In the second scenario described above, a client communicating with a GPL-licensed server will use a protocol that is not specific to that server and not originally created by the developer of that server. Since the protocol and data structures in this scenario have been created by a

---

expected functionality for the applicable API, which can usually be done with independently developed code.

258. An example of this type of situation would be the development of a GPL-licensed IMAP server for use in an email system. (IMAP stands for Internet Message Access Protocol and this protocol is defined in RFC 2060 and RFC 3501). The IMAP protocol supports message handling operations such as, "create", "retrieve", "delete", etc. The IMAP protocol also supports various mail folder management capabilities. The IMAP protocol allows an email client to communicate with an IMAP server to remotely access messages stored on that server.

third party, they will not be original when copied and used in a GPL-licensed server. As with the first scenario, the use of such a third-party protocol and associated data structures probably will be indirectly reflected in the object code of the client program. However, in this case, any non-literal copying will not be copy protected elements of the GPL-licensed server. Instead, it will copy non-literal elements of third-party work. Therefore, the copyright and licensing terms for that third-party work will be critical when assessing the significance of this material as embodied in the client program. In general, if the protocol being used is a well-known protocol, such as Open Database Connectivity ("ODBC") or SMTP, then the license terms for that protocol are not going to be GPL-based. Therefore, in this situation, the primary determinant when assessing any GPL effects is not the scope of copyright protection for data structures and protocols, but is instead their originality when used in the context of a GPL-licensed server. One final issue that will be critical in assessing the GPL effects in both scenarios is whether client-server interactions are methods of operation.

III. AN EXAMINATION OF THE VIRAL PROVISIONS OF THE GPL

The FSF claims that the GPL is a bare copyright license.[259] By this, the FSF means that the legal foundation for the GPL is copyright law and not contract law. Accordingly, the enforceability of the GPL is not based on offer, acceptance, consideration, or any of the other requirements for the creation of a contract.[260] Instead, the enforcement of the

259. *See* Eben Moglen, Enforcing the GNU GPL, http://www.gnu.org/philosophy/enforcing-gpl.html (last visited Apr. 26, 2009). *See also*, Letter from Eben Moglen, General Counsel, Free Software Foundation, to Carol A. Kunze (Oct. 23, 2001), http://www.nccusl.org/nccusl/meetings/UCITA_Materials/kunze-ucita.pdf. In this letter, Eblen Moglen, in his capacity as General Counsel for the FSF, wrote the following about the GPL when answering a question about the applicability of UCITA to the GPL:

> The legal effect of the GPL is therefore to apply only copyright doctrine, purely unilaterally and permissively, to the distribution of software. The GPL, unlike a contract, functions identically in all legal systems observing the fundamental principles of copyright harmonized by the Berne Convention. Because our software may be cooperatively produced by individuals in many different countries, and then redistributed by users everywhere in the world, it is our intention to avoid the formation of contractual relations that would impede the international uniformity of our arrangements.
>
> For both theoretical and practical reasons, therefore, the FSF and all other developers choosing to use the GPL for release of software (a worldwide community of tens of thousands of programmers involved in the creation of thousands of free software programs) specifically do not intend the GPL to be a contractual relationship predicted on mutual assent to obligations. *Id.*

260. This has been a much debated issue. The fact that the FSF has taken the position that the GPL is a bare copyright license is not determinative. To date, most analyses have treated the GPL as a contract. However, most of these analyses have not considered the

GPL is based solely on copyright law principles. This approach has not yet been tested in any United States courts; however, if this rationale is accepted, there are some potential side effects with respect to the purportedly viral provisions of the GPL.[261]

The first side effect of a finding that the GPL is a bare copyright license relates to remedies. It has been suggested that if the GPL is a bare copyright license, then there is no basis upon which a GPL violation can be used to force the licensing of proprietary software under the GPL.[262] The reason for this is that only copyright infringement remedies are available for a failure to comply with the conditions for the grant of a license under the GPL. Accordingly, under this theory the remedies available for a GPL violation should include damages and injunctive relief, but should not include contractual remedies, and particularly should

---

contract versus bare copyright license issue. *See e.g.*, González, *supra* note 8, at 331, 333; González, *supra* note 8, at 331, 333; Wacha, *supra* note 11, at 20, 22-23 (all containing enforceability analyses based on contract principles). *See also* Jason B. Wacha, *Taking the Case: Is the GPL Enforceable?*, 21 SANTA CLARA COMPUTER & HIGH TECH. L.J. 451, 456 (2005) (stating that:

> The GPL is not just a method for a licensor to give up rights that he could otherwise enforce in court; the GPL imposes obligations on the licensee as well, which the licensee must accept. It is likely that a court, in the U.S. or abroad, would recognize the GPL as a contract.) (footnote omitted).

Paul H. Arne & John C. Yates, *Open Source Software Licenses: Perspectives of the End User and the Software Developer*, 22 THE COMPUTER & INTERNET LAWYER 1, 2-3 (Aug. 2005) (acknowledging the license-only position, but observing:

> Most open source licenses contain other provisions that are harder to characterize as some exclusive right that the copyright holder is retaining rather than giving away. Examples of these include limitations of liability and disclaimers of implied warranties, specifically the warranties of merchantability and fitness for purpose implied in some contracts under the Uniform Commercial Code. It is harder to argue that a limitation of liability is a limitation on the ability to make copies, create derivative works, or distribute the software. Accordingly, in order to be enforceable, other laws must probably be relied on other than copyright in order to render them enforceable. Contract law is the likely alternative.) (footnote omitted).

See contra Raymond Nimmer, *Is the GPL license a contract? The wrong question*, CONTEMPORARY INTELLECTUAL PROPERTY, LICENSING & INFORMATION LAW, Sept. 6, 2005, http://www.ipinfoblog.com/archives/licensing-law-issues-30-is-the-gpl-license-a-contract-the-wrong-question.html (taking the position that a determination about whether the GPL is a license or a contract depends on the context of the particular transaction). Pamela Jones, *The GPL Is a License, Not a Contract, Which is Why the Sky Isn't Falling*, GROKLAW, Dec. 14, 2003, http://www.groklaw.net/article.php?story=20031214210634851 (taking the position that the GPL is a license and not a contract). Still other commentators believe that even if the GPL is held to be a contact, it is unlikely a court will grant the remedy of specific performance, *see* Richard A. Epstein, *Why Open Source is Unsustainable*, FINANCIAL TIMES, Oct. 21, 2004, http://www.ft.com/cms/s/78d9812a-2386-11d9-aee5-00000e2511c8.html.

261. *See* Jacobsen v. Katzer, 535 F.3d 1373 (Fed. Cir. 2008) (dealing with the construction of certain conditions in the Artistic License, which may be relevant in construing conditions in the GPL).

262. *See* Jones, *supra* note 260.

not include specific performance to compel the licensing of derivative or infringing works. If this analysis is correct, then the GPL is not viral because a licensor cannot force derivative or infringing works to be licensed under the terms of the GPL and such licensing can only occur if the accused infringer chooses to do so.[263] In the worst case scenario, an accused infringer would be subject to an injunction, damages, and an obligation to pay attorney's fees. For a commercial software vendor, this "worst case" would be bad, but not as bad as potentially losing its future license revenue for the affected product. The affected product would have to be taken out of commercial distribution for a period of time while the infringing or derivative code was removed; however, once this code was removed, the developer could continue to license the product. There are companies for whom such costs and delays might be catastrophic, but it seems reasonable to expect that the majority of commercial software vendors would be able to survive such a scenario.

Unfortunately, the bare copyright analysis set forth above is not complete since it does not take into account the potential effect of §103(a) of the Copyright Act, which provides that:

> The subject matter of copyright as specified by section 102 includes compilations and derivative works, but protection for a work employing preexisting material in which copyright subsists does not extend to any part of the work in which such material has been used unlawfully.[264]

In the circumstances described above, a commercial software vendor that fails to comply with §5(c) of the GPL.v3 (or §2(b) of the GPL.v2) will be unlicensed and thus potentially infringe copyright in the GPL-licensed work. If the commercial software vendor has created a derivative work of the GPL-licensed work then §103(a) should apply to that derivative work. Therefore, even if the copyright holder of a GPL-licensed work cannot compel a commercial software vendor to license its product under the GPL, the software vendor may not be able to use copyright to protect those portions of a derivative work in which GPL-licensed material has

263. *Id.* The following argument has been made in support of this proposition:

> So when you hear that the GPL is viral and can force proprietary code to become GPL, which a couple of lawyers have been saying, you'll know that isn't true. If you steal GPL code, you can expect enforcement, if the violation isn't cured, but it can only be enforcement of a license, not a contract, and a forced release under the GPL can't be imposed on you under copyright law. It's not one of the choices, as Professor Moglen has explained. You do have a choice under the GPL license: you can stop using the stolen code and write your own, or you can decide you'd rather release under the GPL. But the choice is yours. If you say, I choose neither, then the court can impose an injunction to stop you from further distribution, but it won't order your code released under the GPL. This is because under copyright law, as Professor Moglen explained, your penalty is the injunction, damages, and maybe attorney's fees. Your code remains yours, as you can see, even in a worst case scenario. *Id.*

264. 17 U.S.C. § 103(a) (2010) (emphasis added).

been used.[265] It has even been suggested that applying §103(a) means that no one has copyright in the affected portions of the derivative work and those portions fall into the public domain and are theoretically free for anyone to use, copy and/or distribute.[266] Section 103(a) poses an even greater risk in cases where the infringing material pervades a resulting derivative work. In such cases copyright may not apply to any part of such a derivative work.[267] Therefore, while it may be correct to say that the GPL does not have viral effects under a bare copyright license theory, this does not mean that use of GPL-licensed software in violation of §5(c) of the GPL.v3 (or §2(b) of the GPL.v2) cannot have significant intellectual property consequences. Accordingly, the real worst case for a commercial software vendor under a bare copyright license theory appears to be the possibility of injunctive relief, damages (potentially including attorney's fees), and loss of copyright protection for products that include infringing GPL-licensed material. This means a commercial software vendor can potentially lose its ability to prevent unauthorized copying, use or distribution of source or binary versions of its affected products that have been provided to licensees. If the source code for a product has fallen into the public domain through the operation of §103(a), then a recipient of that source code could make it widely available without the software vendor having any recourse under copyright law.[268] If only binary code versions of the affected products have been distributed to customers, then these binary code versions can also be made widely available (a somewhat less destructive scenario). While the likelihood of either of these scenarios seems relatively low, both are potentially far more destructive than the worst case scenario generally suggested in the literature for a violation of the GPL under a bare copyright license theory.

---

265. Sean Hogle, *Unauthorized Derivative Source Code*, 18 THE COMPUTER & INTERNET LAWYER 1, 3 (May 2001).

266. *Id.* at 5.

267. *See* Eden Toys, Inc. v. Florelee Undergarment Co., Inc., 697 F.2d 27, 34 n.6 (2d Cir. 1982) (stating that copyright protection does not extend to derivative works if the pre-existing work tends to pervade the entire derivative work). *See also* Pickett v. Prince, 207 F.3d 402 (7th Cir. 2000) (denying copyright protection where a copyrighted symbol pervaded a derivative work). *See also* Anderson v. Stallone, No. 87-0592, 1989 WL 206431 (C.D. Cal. Apr. 25, 1989) (concluding that copyright protection for unauthorized works is limited to compilations and cannot reasonably be applied to derivative works).

268. It seems unlikely a licensee would take such action since it would almost certainly trigger a lawsuit whose outcome would not be a forgone conclusion. This scenario is probably even more remote since a commercial software developer may have other remedies under trade secret law, patent law, or pursuant to contract law under the developer's license agreement with its licensees. However, a commercial software developer's most potent legal weapon, copyright law, may have been compromised through the operation of §103(a).

The second side effect of a finding that the GPL is a bare copyright license relates to the type of material governed by such a license. If copyright law is the sole basis upon which the GPL operates, then subject matter not covered by copyright law, such as processes, procedures, methods of operation, and other material excluded by doctrines such as merger and scènes à faire, cannot be controlled using the GPL. This limitation is particularly noteworthy since under *ProCD, Inc. v. Zeidenberg*, contract law can be used to protect non-copyrightable portions of software programs.[269] Under copyright law, material in a GPL-licensed work that is not copyright protected may be used in another work without making that second work a derivative or infringing work of the GPL-licensed work. If unprotected material is used from a GPL-licensed work, then the operation of §5(c) of the GPL.v3 should be nullified. The resulting work should not be an infringing or derivative work; and therefore, neither damages nor injunctive relief should be available. Additionally, since no material has been used unlawfully, §103(a) should not apply and copyright in the resulting work should not be compromised. A similar outcome is also possible if the GPL is found to be a contract. In this scenario, the §5(c) requirement (or in the case of the GPL.v2, the §2(b) requirement) would need to be construed as only applying to derivative or infringing works. In the case of the GPL.v2, a variety of terms are used to characterize the scope of the viral provisions. This makes construction of the viral provisions of the GPL.v2 challenging. However, the GPL.v3 seems to make it clear that certain terminology used in the GPL is intended to have the same scope as the term "derivative work" under copyright law. In particular, the terminology used in the GPL that relates to the concept of a work based on another work has been clarified to mean a modified version of a first work.[270] Within that definition, to "modify" means "to copy from or adapt all or part of the work *in a fashion requiring copyright permission*. . . .[t]he resulting work is called a 'modified version' of the earlier work or a 'work based' on the earlier work."[271] Additionally, to "propagate" a work has been similarly defined as "do[ing] anything with it that, without permission, *would make you directly or secondarily liable for infringement under applicable copyright law*."[272] Finally, the term "Program" as used in the GPL.v3 is defined as "*any copyrightable work* licensed under this License,"[273] presumably meaning that any material not covered by copyright is not subject to the GPL.

269. ProCD, Inc. v. Zeidenberg, 86 F.3d 1447 (7th Cir. 1996).

270. Free Software Foundation, *GNU General Public License, Version 3,* §0, June 29, 2007, http://www.gnu.org/licenses/gpl-3.0.txt.

271. *Id*. (emphasis added).

272. *Id.* (emphasis added).

273. *Id.* (emphasis added).

In the remainder of this paper it will be assumed that the GPL is a bare copyright license and the analysis of the scope of the viral provisions of the GPL will be done on that basis. However, as discussed above, similar results can arise under a copyright analysis if the GPL is construed to cover only material protectable under copyright. When discussing the GPL, any references to the viral provisions of the GPL or the viral nature of that license should be understood to encompass the entire spectrum of negative consequences associated with a failure to comply with the conditions of §5(c) (or 2(b) in the case of the GPL.v2). These negative consequences include: industry or open source community pressure to designate a commercial product as free software, damages, injunctive relief, and loss of copyright protection in a work that unlawfully uses GPL-licensed material.

A. The GPL License

The viral provisions of the GPL.v3 are primarily contained in §5 and §6 and rely on definitions in §0 and §1.[274] These provisions and definitions are as follows:

---

274. *Id. See also* Free Software Foundation, GNU General Public License, Version 2, June 2, 1991, http://www.gnu.org/licenses/gpl-2.0.txt (for the corresponding provisions from the GPL.v2:

> 0. This 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. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".
>
> 2. 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 work under the terms of Section I above, provided that you also meet all of these conditions: . . .
>
> (b) You must cause any work that you distribute or publish, 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. . . .
>
> These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
>
> 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 based on the Program.
>
> 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).

A "covered work" means either the unmodified Program or a work based on the Program.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License . . .

5. You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions: . . .

> c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what

> the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

The viral effects of the GPL.v3 depend on the scope of the defined term "covered work." The term "covered work" depends on the scope of the terms "Program" and "work based on the Program." The scope of the term "Program" is relatively straightforward; it is simply the copyrightable portions of the software in which a copyright holder has placed a notice stating that the applicable program is being licensed under the GPL. The term "a work based on the Program" is not defined directly, but is given meaning in the definition of "modify." Particularly, that definition specifies that a work resulting from any copying or adaptation of all or part of an earlier work in a fashion requiring copyright permission, other than the making of an exact copy, is a "work based on the earlier work." This corresponds to the generally understood meaning of a "derivative work" under copyright law. This interpretation is supported by the definition of "propagate," which is "[anything that] would make you directly or secondarily liable for infringement under applicable copyright law."

However, there is other language in the GPL.v3 that seems to go beyond the traditional scope of a "derivative work." Specifically, one of the conditions that the licensee must meet before conveying a binary version of a "covered work" is that the licensee must also provide a copy of the "corresponding source." The definition of "corresponding source" purports to include "interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work." This language, which was newly introduced in the GPL.v3, is similar to the scope of viral coverage previously claimed by the FSF for the GPLv2. As will be discussed later, the material that needs to be incorporated into a work to allow it to dynamically link to another program or to communicate with another program via inter-process communication does not necessarily make a calling program a derivative work of a called program. Additionally, a program that is dynamically called by another program generally will not be a derivative work of the calling program. The foregoing is also generally true for programs that communicate by inter-process communication. Accordingly, to the extent the GPL.v3 is characterizing these materials as parts of a covered work, this characterization appears to go beyond copyright law's established bounds, and hence makes some parts of the GPL.v3 internally inconsistent.

The definition of a work based on the program in the GPL.v2 is similar to that in the GPL.v3. In the GPL.v2 that term is described as either

the Program or any derivative work under copyright law. As with the GPL.v3, the GPL.v2 contains other language that potentially encompasses subject matter beyond a derivative work under copyright law. Specifically, the GPL.v2 states that a derivative work is "a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language." Typically, a program will contain copyrightable subject matter.[275] However, most programs also will contain subject matter that is not protected by copyright. As a matter of copyright law, a subsequent work cannot be a derivative or infringing work of a prior work unless it has substantially copied protected subject matter from that prior work.[276] Accordingly, to the extent that the GPL.v2 suggests that the copying of any subject matter from "the Program" necessarily makes a subsequent work a derivative work, that statement is incorrect. In addition to the foregoing language, §2 of the GPL also contains language relating to derivative works. The language in §2 states that it is not the intent of the GPL.v2 to claim or contest rights in works written entirely by the licensee, but instead "to control the distribution of derivative or collective works based on the Program." If the GPL.v2 is only a bare copyright license, then the only way this provision can be interpreted consistently with the earlier provisions dealing with works "based on the Program" is if that term is intended to mean works that have substantially copied protected subject matter from the Program.

Both the GPL.v3 and the GPL.v2 contain exemptions for "aggregations" and "mere aggregations." In the GPL.v3, this exemption excludes other "separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program." This language appears to diverge from other language in the GPL.v3 such as the definitions of "to modify" and "to propagate" which are cast in terms of copyright concepts. In particular, this language is potentially inconsistent with other parts of the GPL.v3 because programs can be extended or combined without copying or adapting copyrightable subject matter. The corresponding language in the GPL.v2 raises similar concerns. The GPL.v2 provides that "a work not based on the program," can be aggregated on a storage or distribution medium with a Program (or a work based on a Program) without the "work not based on the Program" being made subject to the GPL.v2. Unfortunately, this language is difficult to construe because it does not use standard copyright terminology. However, if this provision is to be

275. *See contra* Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522 (6th Cir. 2004) (holding that none of Lexmark's Toner Loading Program was eligible for copyright protection).

276. *See*, MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT §3.01 3-4 (Release No. 70, June 2006).

interpreted in a manner consistent with copyright law and other parts of the GPL, then the phrase "not based on" needs to be read as "not substantially copying protected subject matter from," which, under copyright law, will mean that such a work is not a derivative work.

As discussed above, the "viral" portions of both the GPL.v2 and the GPL.v3 contain language that is potentially inconsistent. In order for the GPL to be interpreted in a consistent manner and in order for the GPL to cover subject matter that is consistent with its characterization as a bare copyright license, some of the language in the GPL needs to be construed as corresponding to the definition of derivative work and the related right under copyright law. All of the foregoing means that a very detailed understanding of derivative works is critical to an understanding of the viral effects of the GPL. The importance of such an understanding was confirmed in *Progress Software Corp. v. MYSQL AB*.[277] In the context of a request for a preliminary injunction, the court in that case stated that "[a]ffidavits submitted by the parties' experts raise a factual dispute concerning whether the Gemini program is a derivative work or an independent and separate work under GPL ¶2."[278] The GPL.v2 was also considered in *Computer Associates Int'l., Inc. v. Quest Software, Inc*.[279] However, the *Computer Associates* case did not examine the application of §2 of the GPL.v2. Instead, the GPL-related aspects of that case concerned a special exception granted by the FSF with respect to certain files copied into an output file used to create the program at issue.

B. Derivative Works

The Copyright Act defines a "Derivative Work" as:

> [A] work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, 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 of authorship, is a "derivative work."[280]

The first thing to notice about this definition is that it requires a pre-existing work to have been recast, transformed or adapted in some manner.[281] As described in *Lee v. A.R.T. Co.*, to be considered a deriva-

277. Progress Software Corp. v. MySQL AB, 195 F. Supp. 2d 328 (D. Mass. 2002).

278. *Id*. at 329.

279. Computer Assoc. Int'l, Inc. v. Quest Software, Inc., 333 F.Supp. 2d 688, 697-98 (N.D. Ill. 2004).

280. 17 U.S.C. §101 (2010).

281. SHL Imaging, Inc. v. Artisan House, Inc., 117 F. Supp. 2d 301, 306 (S.D.N.Y. 2000) ("any derivative work must recast, transform or adopt [sic] the authorship contained in the

tive work, a work must qualify as one of the enumerated types of work (i.e. a translation, dramatization, fictionalization, or any of the other specifically identified categories), or qualify as "any other form in which a work may be recast, transformed, or adapted."[282]

The second point to notice about the definition of "derivative work" is that it is very broad. In fact, the statutory language has been described as "hopelessly overbroad"[283] and it has been further observed that almost all works are derived from pre-existing works.[284] A very broad scope of protection that does not support the fundamental goals of copyright law becomes possible when the breadth of the derivative works' right is coupled with copyright protection against both literal and non-literal copying, applied to a scientific discipline in which development occurs in an incremental manner, and where interoperability and standardization are critical.[285]

Unsurprisingly, the risk posed by a broad interpretation of the derivative works right has been the subject of much comment in academic literature.[286] For example, *Micro Star v. Formgen Inc*., is cited as authority for the proposition that a work can be considered a derivative

---

preexisting work"). *See* NIMMER & NIMMER, *supra* note 276, at §3.03 3-10 ("A derivative work consists of a contribution of original material to a pre-existing work so as to recast, transform or adapt the pre-existing work").

282. Lee v. A.R.T. Co., 125 F.3d 580, 582 (7th Cir. 1997).

283. Micro Star v. Formgen Inc., 154 F.3d 1107, 1110 (9th Cir. 1998).

284. Emerson v. Davies, 8 F. Cas. 615, 619 (C.C.D. Mass. 1845). In this case, Justice Storey observed:

> In truth, in literature, in science and in art, there are, and can be, few, if any, things which, in an abstract sense, are strictly new and original throughout. Every book in literature, science and art, borrows and must necessarily borrow, and use much which was well known and used before. . . . If no book could be the subject of copyright which was not new and original in the elements of which it is composed, there could be no ground for any copyright in modern times, and we would be obliged to ascend very high, even in antiquity, to find a work entitled to such eminence.

285. U.S. CONST. art. I, §8, cl. 8 (providing that Congress has the "Power To promote the Progress of Science by securing for limited Time to Authors . . . the exclusive Right to Their Writings"). The primary goal of U.S. copyright law is not to compensate or reward authors, but to promote the progress of science and the useful arts. Accordingly, the rights granted to authors are a means, not an end, and the rights granted to authors under copyright must be interpreted in light of the overall constitutional goal.

286. *See generally* Edward G. Black & Michael H. Page, *Add-On Infringements: When Computer Add-Ons and Peripherals Should (and Should Not) Be Considered Infringing Derivative Works Under Lewis Galoob Toys, Inc. v. Nintendo of America, Inc., and Other Recent Decisions*, 15 HASTINGS COMM. & ENT. L.J. 615 (1993); Michael Gemignani, *Copyright Protection: Computer-Related Dependent Works*, 15 RUTGERS COMPUTER & TECH. L.J. 383 (1989); Hogle, *supra* note 265; Lydia Pallas Loren, *The Changing Nature of Derivative Works in the Face of New Technologies*, 4 J. SMALL & EMERGING BUS. L. 57 (2000); Christian H. Nadan, *A Proposal to Recognize Component Works: How a Teddy Bears on the Competing Ends of Copyright Law*, 8 CAL. L. REV. 1633 (1990).

work even if it lacks incorporated expression.[287] In *Micro Star*, the manufacturer and distributor of the video game Duke Nukem 3D objected to the distribution of a collection of user created game levels.[288] The alleged infringer, Micro Star, gathered approximately 300 of these user-created levels and distributed them on a Compact Disk.[289] Formgen objected to this distribution and Micro Star sought a declaratory judgment that it was not infringing any of Formgen's copyrights.[290] In deciding the request for a preliminary injunction, the Ninth Circuit Court of Appeals ruled that Formgen had shown it was likely to succeed at trial in establishing that Micro Star had infringed Formgen's right to prepare derivative works.[291] Despite the academic commentary, a close reading of *Micro Star* suggests that the case supports the proposition that a derivative work is created when non-literal expression is copied from a fictional work, rather than a case that supports the proposition that copying incorporated expression is not required to create a derivative work.[292]

In *Sega Enterprises Ltd. v. Accolade, Inc.*, the Ninth Circuit Court of Appeals noted that "[w]orks of fiction receive greater protection than works that have strong factual elements, such as historical or biographical works, or works that have strong functional elements, such as accounting textbooks."[293] The greater protection accorded to fictional works helps explain the apparently broad derivative works protection granted in certain computer-related cases. This difference also provides a basis for concluding that similarly broad protection will not be granted to functional works that dynamically link to or communicate with other functional works. In virtually every case where very broad derivative works protection has been granted to a computer-related work, that computer-related work has had very strong artistic components, such as video game graphics, characters and storylines. It has been these artistic components and not the technical and functional aspects of these works that have attracted strong derivative works protection.

Micro Star is a prime example of how an artistic component of a work can significantly broaden the protection provided to a computer-related work. In that case, the allegedly infringing work was the video

287. Hogle, *supra* note 265; Loren, *supra* note 286, at 73-74.

288. Micro Star v. Formgen Inc. 154 F.3d 1107 (9th Cir. 1998). The Duke Nukem game allowed users to create, play and store their own game levels. *Id*.

289. *Id.*

290. *Id.*

291. *Id.*

292. The Micro Star case can also be viewed as one in which non-literal aspects of the Duke Nukem 3D MAP files were copied in the user-created game levels.

293. Sega Enterprises Ltd. v. Accolade Inc., 977 F.2d 1510, 1524 (9th Cir. 1992) (citations omitted).

game Duke Nukem 3D.[294] The game consisted of three distinct elements: the game engine, the source art library, and the MAP files.[295] The game engine was a computer program that directed the operation of the game.[296] This program instructed the underlying hardware when to read data, play sounds and display images on a video screen.[297] The game engine also provided an ability to save and load games.[298] The source art library was simply a repository of information describing various images used when the game was playing. The MAP files contained information used by the game engine to construct and populate levels in the Duke Nukem 3D game.[299] When the game was started, the game engine would extract images from the source art library for the various components identified in the MAP file for the level being played.[300] These images were then presented on the video display.[301]

In *Micro Star*, the Ninth Circuit Court of Appeals made it very clear that the protected work was the Duke Nukem story and that the non-literal expression copied by the defendant included the characters, settings and plot of the Duke Nukem story.[302] The Ninth Circuit's approach is interesting because the court did not accept Micro Star's defense that the MAP files were not derivative works because they did not incorporate any protected expression from the Duke Nukem pro-

294. Micro Star v. Formgen Inc. 154 F.3d 1107 (9th Cir. 1998).

295. *Id*. at 1110.

296. *Id*.

297. *Id*.

298. *Id*.

299. *Id*.

300. *Micro Star*, 154 F.3d at 1110.

301. *Id*.

302. *Id.* at 1112 (stating:

> In making [its] argument, Micro Star misconstrues the protected work. The work that Micro Star infringes is the D/N-3D story itself—a beefy commando type named Duke who wanders around post-Apocalypse Los Angeles, shooting Pig Cops with a gun, lobbing hand grenades, searching for medkits and steroids, using a jetpack to leap over obstacles, blowing up gas tanks, avoiding radioactive slime. A copyright owner holds the right to create sequels, *see Trust Co. Bank v. MGM/UA Entertainment Co*., 772 F.2d 740 (11th Cir.1985), and the stories told in the N/I MAP files are surely sequels, telling new (though somewhat repetitive) tales of Duke's fabulous adventures. *A book about Duke Nukem would infringe for the same reason, even if it contained no pictures.*) (emphasis added).

The reference to the *MGM* case and the nebulous "right to create sequels" is unfortunate, because that case contains no significant analysis of copyright protection for derivative works. A more thoughtful examination of this issue (in the context of the very same pre-existing work – Gone With The Wind) is provided in Sun Trust Bank v. Houghton Mifflin Co., 252 F.3d 1165 (11th Cir. 2001). In that case, the Eleventh Circuit Court of Appeals found substantial similarity between the pre-existing work Gone With The Wind and the subsequent work The Wind Done Gone. The finding of substantial similarity was based on the use of numerous characters, settings and plot twists from the pre-existing work.

gram.[303] Instead of examining whether Micro Star had infringed the Duke Nukem program, the court focused on the Duke Nukem story. Having made this choice, the conclusion that Micro Star had copied non-literal elements of the Duke Nukem story was relatively easy for the court to reach. However, the outcome may have been different if the court had chosen to examine the case from a more software-based perspective.

Micro Star's defense was based on the fact that the MAP files referenced the source art library but did not contain any actual source art files. While this proposition was undoubtedly true, this fact alone does not necessarily mean Micro Star did not infringe the Duke Nukem program. The MAP files did not contain any source art, but the MAP files were still based on the structure, sequence and organization of the game engine and the source art library, thereby enabling the game engine to properly read those files and identify the parts of the source art library to be displayed for a particular game level. If a software-based approach had been used, the question would still have been one of non-literal infringement. However, the work at issue would not have been a fictional or artistic work, which is entitled to broad copyright protection. Instead, it would have been a functional work, which is entitled to narrower protection. In addition, under a software-based approach, the Ninth Circuit also would have to determine whether the MAP files were part of a method of operation for the Duke Nukem 3D program. By selecting the Duke Nukem story as the initial work and by granting broad copyright protection to the characters, settings and plot of the Duke Nukem story, the Ninth Circuit also granted the copyright holder a *de facto* monopoly over the MAP file format. The *Micro Star* case is probably atypical because the fact that a functional work was overlaid with a fictional work meant that the functional work was granted a level of copyright protection more typical for a fictional work.

For programs linking to or communicating with GPL-licensed works, the type of non-literal copying that may occur will be highly correlated to the function of those programs. In the vast majority of cases these types of linkages and communication mechanisms will not be overlaid with an artistic work that invites broad copyright protection.[304] Accordingly, it

303. In other cases the structure, sequence and organization of input files have been found to be non-literal elements entitled to copyright protection. Accordingly, it could have been argued that Micro Star was copying non-literal elements of the Duke Nukem program.

304. It would be unusual for a case involving linking or inter-process communication to also involve an audio-visual work or any fictional or artistic subject matter. Instead, the circumstances are more likely to be similar to those described in Mitchell Zimmerman, *Baystate: Technical Interfaces Not Copyrightable-On to the First Circuit*, 14 THE COMPUTER LAWYER 9, 9 (Apr., 1997).

appears that the derivative works cases that have been most criticized for providing overly broad protection will be of little relevance when determining whether linking to or communicating with another program makes the linking or communicating program a derivative work. Since works in a case involving linking or inter-process communication will be very different from the highly artistic work considered in *Micro Star*, and since the scope of protection to be provided to more functional works is significantly narrower than that provided to highly artistic works, *Micro Star* (and similar cases dealing with highly artistic subject matter) are probably going to be easy to distinguish from linking and inter-process communication cases.[305]

---

305. For example, in Worlds of Wonder, Inc. v. Veritel Learning Sys., Inc., 658 F. Supp. 351 (N.D. Tex. 1986) and Worlds of Wonder, Inc. v. Vector Intercontinental, Inc., 653 F. Supp. 135 (N.D.Ohio 1986), the pre-existing audio-visual work was a Teddy Bear that moved and told a story. In Worlds of Wonder v. Veritel, 658 F.Supp. at 356, the court said the subsequent work "creates a substantially similar *audiovisual work* which is altered in much the same as [sic] a Galaxian game is altered by a speed up kit" (emphasis added). In Worlds of Wonder v. Vector, 653 F.Supp. at 139, the court described its substantial similarity analysis as follows, "During the hearing, the Court compared the work created when a W.O.W. cassette activated Teddy Ruxpin and the work created when a Vector tape activated Teddy Ruxpin." The court noted a number of similarities such as the use of a male voice with a similar tone, pitch and pace as well as the visual impression of the eyes, nose and mouth movement. *Id.* The court then observes that "the general feel and concept of Teddy Ruxpin when telling a fairy tale is the same regardless of whether a W.O.W. or a Vector tape is used; the visual affects [sic] are identical, and the voices are similar, and the difference in stories does not alter the aesthetic appeal." *Id.* at 140. In Midway Mfg. Co. v. Artic Int'l, Inc., 704 F.2d 1009 (7th Cir. 1983), the pre-existing works are once again audio-visual works, in this case, speeded-up versions of the video games Galaxian and Pac-Man. In *Midway*, the court concluded that the speeded-up Galaxian and Pac-Man games were derivative works. *Id.* While the approach taken by the court in making this determination is somewhat different than the usual substantial similarity analysis, it was primarily based on the fictional/artistic aspects of the pre-existing work rather than its software characteristics. In particular, the court said:

> It is not obvious from [the definition of a derivative work] whether a speeded-up video game is a derivative work. A speeded-up phonograph record probably is not. But that is because the additional value to the copyright owner of having the right to market separately the speeded-up version of the recorded performance is too trivial to warrant legal protection for that right. A speeded-up video game is a substantially different product from the original game. As noted, it is more exciting to play and it requires some creative effort to produce. For that reason, the owner of the copyright on the game should be entitled to monopolize it on the same theory that he is entitled to monopolize the derivative works specifically listed in Section 101. . . . But the amount by which the language of Section 101 must be stretched to accommodate speeded-up video games is, we believe, within the limits within which Congress wanted the new Act to operate. (citations omitted) *Id*. at 1014.

Another line of derivative works cases dealing with the mounting of note cards, lithographs, and cutouts from art books onto ceramic tiles: Mirage Editions, Inc. v. Albuquerque A.R.T. Co., 856 F.2d 1341 (9th Cir. 1988), Munoz v. Albuquerque A.R.T. Co., 829 F. Supp. 309 (D.Alaska 1993), Lee v. A.R.T. Co., 125 F.3d 580 (7th Cir. 1997), can be similarly distinguished from linking and inter-process communication on the basis that these cases

Once the cases dealing with audiovisual works or other artistic works have been distinguished, a derivative works analysis in a software setting becomes a matter of applying a number of well-known copyright principles. Admittedly, these well-known principles may have to be applied in technical areas that have not received significant judicial consideration, but at least there is a reasonable basis for predicting likely outcomes.

The first principle in a derivative works analysis is that a work cannot be a derivative work unless it has substantially copied protected material from a prior work.[306] Phrased another way, a work will not be a derivative work unless it is also an infringing work because of the material copied from the pre-existing work.[307] Due to this limitation, the term derivative work does not include all works that borrow to some degree from pre-existing works.[308] If a subsequent work copies material from a pre-existing work that is not subject to copyright protection, then that subsequent work cannot be a derivative work of the pre-existing work.

The determination of whether one work is substantially similar to another work is not limited to literal copying. It is a well-established copyright principle that the copying of non-literal material from a pre-existing work can satisfy the substantial similarity requirement and make a subsequent work a derivative work. This principle is significant because it means that a work can become a derivative work by copying the structure, sequence and organization of another work.[309] Therefore, cases like *Computer Associates International, Inc. v. Altai, Inc.*,[310] *Gates Rubber Co. v. Bando Chemical Industries, Ltd.*[311] and the numerous other cases dealing with non-literal copying are very significant in conducting a derivative works analysis. The use of the substantial similarity standard also means that various concepts, such as: merger; scènes à faire; exclusion of copyright protection for ideas, procedures, processes,

---

once again deal with artistic works rather than functional works. In these cases, even when applying the artistic standard of protection, two circuit courts of appeal have expressed conflicting views about whether the mounting of pre-existing art onto ceramic tiles results in the creation of a derivative work.

306. Micro Star v. Formgen, Inc., 154 F.3d 1107, 1110 (9th Cir. 1998); Vault Corp. v. Quaid Software Ltd., 847 F.2d 255, 267 (5th Cir. 1988); Litchfield v. Spielberg, 736 F.2d 1352, 1357 (9th Cir. 1984); SHL Imaging, Inc. v. Artisan House, Inc., 117 F. Supp.2d 301, 305 (S.D.N.Y. 2000); NIMMER & NIMMER, *supra* note 276, at §3.01 3-4.

307. NIMMER & NIMMER, *supra* note 276, at §3.01 3-4.

308. *Id*.

309. Dan Ravicher, *Software Derivative Work: A Circuit Dependent Determination*, GROKLAW, Nov. 8, 2002, http://www.groklaw.net/pdf/ravicher_1.pdf.

310. Computer Assoc. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992).

311. Gates Rubber Co. v. Bando Chemical Indus., Ltd., 9 F.3d 823 (10th Cir. 1993).

systems, and methods of operation; and the requirements for originality are also very significant.

Therefore, a derivative works analysis of linking or inter-process communication with a GPL-licensed program will involve an assessment of whether that program has substantially copied protectable subject matter from the GPL-licensed program. The first step in this process will be to identify those portions of the GPL-licensed program that are copied by a linking or communicating program. The next step will be to determine whether the copied material has been transformed, recast or altered in the calling or communicating work. If this has not happened, then the calling or linking program cannot be a derivative work. The calling or linking program may infringe other rights protected under copyright, such as the reproduction right, but it cannot infringe the derivative work right.[312]The material copied from a GPL-licensed program must also be examined to determine whether it is protectable subject matter under copyright. Once all protectable subject matter has been identified, that material needs to be assessed to determine whether it is a substantial part of the GPL-licensed program. Finally, when conducting this analysis it is important to remember that copying may be both literal and non-literal.

C. GPL "Viral" Effects and Static Linking

This section will examine whether static linking to a GPL-licensed library will make the linking program a derivative or infringing work of the linked library and thereby engage the viral provisions of the GPL. The first step in the analysis of static linking is to identify the copied material. As described in § II.B, a statically linked ELF object file contains various symbol names gathered from the program itself and from any linked libraries. When a calling program statically links to a GPL-licensed library, the symbol names in the GPL-licensed library that are referenced by the calling program will be copied into its object code file. In addition, a statically linked program will also contain object code for any routines called from any linked libraries.[313] A calling program may

312. Under the GPL.v2 this distinction may be significant since the viral provisions of the GPL.v2 appear to only apply if a derivative work has been created. Given the otherwise broad licenses in the GPL, it is conceivable that under the GPL.v2 a work that copies a portion of a GPL-licensed work, without becoming a derivative work may be distributed without engaging any viral obligations. This loophole, if it exists, appears to have been closed in the GPL.v3, which uses broader terminology that is not limited to derivative works.

313. This assumes that the calling program is invoking a function or procedure. While a calling program may link to a library simply to obtain access to a particular data structure definition or global variable, the much more common case will involve a calling program that invokes one or more functions or procedures in a linked library.

also copy other non-literal elements from linked libraries. For example, to the extent a calling program uses non-literal aspects of a library, such as data structures or other structure, sequence or organization, these elements can also support a finding of substantial similarity. That being said, in all but rare circumstances, the verbatim copying of object code from a linked library will be the determinative factor in deciding whether a calling program is a derivative or infringing work of a statically linked library.

Having identified the copied material, the next step in the derivative works analysis is to determine whether that material has been recast, transformed or adapted. According to The Merriam-Webster Online Dictionary, "adapt" means "to make fit (as for a specific or new use or situation) often by modification."[314] Object code in a library is in a form that allows that code to be included in other programs. Object code that has been statically linked into a program is in a form that allows that computer to execute a code. Accordingly, it seems reasonably certain that the transformative requirement has been met since the copied material has been taken from a setting in which it can be included in programs and it has been put in a setting in which a computer can execute it.[315]

The final step in the analysis is to determine whether the copied material represents a substantial copying of the pre-existing work. This is a qualitative rather than a purely quantitative analysis.[316] Therefore, the assessment is not simply done on the basis of how much material is copied, but instead involves an examination of the significance or importance of the copied material when compared to the work as a whole.[317] This assessment is necessarily fact dependent and will vary according to the particular functions and routines invoked by a calling program and their relative significance in the context of the called library as a whole. That being said, it seems reasonable to assume that most commercial uses of a statically linked GPL-licensed library will result in the copying of a qualitatively significant portion of the library. If this was not the case, there would be little reason to use the library since a trivial amount of code, or code that is trivial to implement, can be easily written by a developer without incurring the risk of any GPL viral effects.[318] In most

314. Merriam-Webster Online Dictionary, Adapt, http://www.m-w.com/dictionary/adapt (last visited Apr. 26, 2009).

315. In addition to changing the purpose for which the code is used, the structure surrounding the program is significantly different and the manner in which the code is accessed is also quite different.

316. Gates Rubber Co. v. Bando Chemical Indus., Ltd., 9 F.3d 823, 839 (10th Cir. 1993).

317. *Id.* at 839-40 n.15.

318. By writing such code the developer will avoid any GPL effects. Given the limited benefits of using a trivial or minor amount of functionality from a GPL-licensed library, the associated risk does not seem justifiable.

cases, a GPL-licensed library is going to be used because the library contains complex, high-quality code that has been tested through broad use. If the code being used is *de minimus*, the benefits of using open source software are not significant.[319]

1. *A Comparison with the Free Software Foundation Position on Static Linking*

The FSF has said that "if the modules are included in the same executable file, they are definitely combined in one program."[320] By this statement, the FSF is saying that static linking engages the viral provisions of the GPL.

While most static linking will make a calling program a derivative or infringing work, a definitive determination will necessarily depend on the actual amount and type of material copied in each case. A derivative or infringing work is less likely to be created when the number of procedures or functions used from a statically linked library is very low, when those functions and procedures are relatively small, and when the bulk of the material in those functions and procedures can be excluded from copyright protection under one or more limiting doctrines.[321] If a calling program is statically linked to routines containing no copyrightable subject matter, then the calling program cannot become a derivative or infringing work of the called library and cannot engage the viral provisions of the GPL.[322]

---

319. A small amount of code can be written and tested in a reasonably short period of time; accordingly there should not be any significant time-to-market benefit from using a small amount of open source. Additionally, if the code in question is straightforward, it is unlikely there will be any significant quality issues. In such cases, a developer should be able to write the required code correctly because it will not be complicated. The benefit of having thousands of developers reviewing such limited amounts of code should not significantly improve its quality.

320. Free Software Foundation, Frequently Asked Questions About Version 2 of the GNU GPL, http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html (last visited Aug. 14, 2010).

321. *See* Gates Rubber Co. v. Bando Chemical Indus., Ltd., 9 F.3d 823, 836-38 (10th Cir. 1993) (describing these limiting doctrines). Among the limitations listed in that case are (a) the idea-expression dichotomy, (b) the process-expression dichotomy, (c) facts, (d) public domain, (e) the merger doctrine, and (f) scènes à faire. *Id.* In Computer Assoc. Int'l, Inc. v. Altai, Inc., the court identified the following types of unprotected subject matter: elements dictated by efficiency, elements dictated by external factors, and elements taken from the public domain. Computer Assoc. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 707-10 (2nd Cir. 1992).

322. Such a scenario is more likely if the particular function or procedure implements a well-known protocol that does not originate from the writer of the particular library. In such a case there may be so little copyrightable material that even verbatim copying will not render a calling program a derivative work.

The foregoing analysis largely agrees with the FSF position and the large majority of writers who have considered this issue. However, this analysis differs from the FSF position to the extent that it acknowledges the possibility that there are certain limited scenarios in which static linking to a GPL-licensed library does not necessarily make a calling program a derivative or infringing work of that library.[323]

2. *Conclusion – GPL "Viral" Effects and Static Linking*

Virtually all commercial-level instances of static linking to a GPL-licensed library are going to cause the calling program to become a derivative or infringing work of the called library. Thus, this will engage the viral provisions of the GPL.

D. GPL "Viral" Effects and Dynamic Linking

In this section dynamic linking to a GPL-licensed library will be examined. In particular, this section will consider the question of whether dynamic linking makes a linking program a derivative or infringing work and thereby engages the viral provisions of the GPL.

As described in Section II.C for an ELF format file, a dynamically linked object file contains various symbol names. As in the case of a statically linked object file, some of these symbol names will be external symbols from any linked GPL-licensed libraries. However, unlike the case of a statically-linked object file, a dynamically linked object file will not contain any object code from any GPL-licensed libraries to which it is being linked. The issue of non-literal copying remains relevant because the use of data structures, protocols, APIs and parameter lists from a called library may result in copying of non-literal elements of that library. Accordingly, the scope of copyright protection accorded to non-literal elements of a software program needs to be examined in detail. Finally, since compilation is a multi-stage process, the various program formats in that process will need to be examined to see if they are derivative works of any libraries to which they are linked and whether those intermediate program files affect the legal status of the final form of the resulting ELF object file.

---

323. Frequently Asked Questions about Version 2 of the GNU GPL, *supra* note 320. The "combined in one program" language is yet another way in which the FSF attempts to describe the criteria that engage the viral provisions of the GPL. *Id.* While the language used in the GNU FAQ is not used in the GPL itself, it is useful for a commercial vendor contemplating GPL software use to consider this language since it is indicative of the FSF's interpretation of the GPL. The FSF also says "Combining two modules means connecting them together so that they form a single larger program. If either part is covered by the GPL, the whole combination must also be released under the GPL — if you can't, or won't, do that, you may not combine them." *Id*.

1. *A Derivative Works Analysis of Dynamic Linking*

Unlike the case with static linking, an analysis of whether dynamic linking engages the viral provisions of the GPL is not straightforward. In fact, many legal scholars have considered this issue and reached differing conclusions.[324] Before considering the more complex issues, this analysis will deal with certain issues that are more easily resolved. The first of these preliminary issues relates to the material to be considered when determining whether a dynamically linked program is a derivative or infringing work of a library to which it is being linked.

The derivative work requirement for the copied material to have been recast, transformed or adapted appears to have been met. When in a dynamically linkable library, the various function, procedure, variable, and data structure definitions are in a form that allows them to be referenced in other programs. Dynamically linking to these function, procedure, variable and data structure definitions causes the storage of data and references and the creation of binary code within a linking program that will allow execution to pass from the linking program to the linked library once both of those programs have been loaded onto a computer.[325] In the case of data structure definitions, the transformation is somewhat less clear. In both a linking program and a linked library, the purpose of a data structure definition is to provide information about the layout of the defined structure and its constituent elements. In each case, the linking program and the linked library are using the data structure definition to understand the actual layout of data in computer memory to allow other parts of a program to access and use the relevant data as intended. Perhaps the only transformative difference is that in the context of a linking program, the data structure is being used in a different program from the one in which it was originally defined. Notwithstanding the possible difficulties in identifying a transformative use for data structure definitions, it still seems reasonably certain that the transformative requirement for a derivative work will be met when consideration is given to all of the material typically copied from a GPL-licensed library for dynamic linking.

Another issue that can be addressed with a reasonable degree of certainty relates to the significance of the various intermediate files created during the compilation process. As discussed in the case law and legal literature, computer programs generally exist in two forms – source code and object code. This, however, is a simplification, since the compilation

324. *See supra* note 11.

325. As described earlier in Section II.C, dynamic linking uses a symbol table, GOT and PLT and various jump and load instructions to cause the absolute addresses of referenced variables or called procedures to be dynamically linked into the object code of a calling program.

process causes the creation of a variety of intermediate files. For dynamically linked programs, the source code file, the assembly language file, and the object file contain equivalent material from a copyright perspective. The source code file will contain one or more *include* statements and various references to global variables, functions, procedures and data structures and their sub-elements. The assembly language file and the object file will contain the names of the global variables, functions, and procedures referenced in the source code file. These files will also contain the names of the libraries to be dynamically linked. Finally, the source, assembly and object code files will all potentially contain non-literal elements from linked object libraries. Accordingly, an analysis of whether the source, object, and assembly language files are derivative works of a linked library requires a consideration of equivalent material.[326]

However, the intermediate file created by the C Preprocessor is different. As described earlier, each *include* statement is a directive to the C preprocessor to cause the *include* statement to be replaced with the entire contents of the file name referred to in that statement. Accordingly, an intermediate file will contain all of the materials from the source code file plus all of the materials contained in any files referenced in any *include* statements in the source code file. In many cases, the use of *include* statements will cause the inclusion of a significant quantity of qualitatively significant copyrighted material into the intermediate file. While each substantial similarity analysis will always be fact dependent, in most cases the inclusion of such a large amount of material is going to be substantial enough for an intermediate file to become a derivative or infringing work of any GPL-licensed libraries to which it is being linked. If intermediate files were to be distributed they would almost certainly engage the viral provisions of the GPL.

Even though an intermediate file is likely to be a derivative work of a library to which it is being linked, it does not appear that the final object file is also necessarily a derivative work. Under the GPL.v3, the viral provisions are triggered by modification and propagation. Each of these terms as used in the GPL is dependent on the occurrence of an act requiring copyright permission or an act giving rise to copyright infringement.[327] Since intermediate files are not normally distributed with com-

326. From a copyright perspective, the inclusion of the file name of a linked object library probably does not change the derivative work analysis significantly.

327. Similarly, under the terms of Section 2(b) of the GPL.v2, "You must cause any work that you distribute or publish, 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." Free Software Foundation, *GNU General Public License, Version 2,* §0, June 29, 2007, http://www.gnu.org/licenses/gpl-3.0.txt.

mercial products,[328] the viral provisions of the GPL are not engaged simply because an intermediate file is created during compilation. The analysis, however, does not end there. One must also consider whether the creation of an intermediate file that is a derivative work of a dynamically linked library consequently means that the final object code file will also be a derivative work. This analysis requires an examination of the relationship between the various forms of a computer program. Since this relationship has not been definitively established, the analysis is speculative.[329][330]

While copyright protection is well established for both the source and object code forms of a computer program,[331] the theoretical basis upon which copyright is extended to the object code form of a computer program and the relationship between the source and the object code forms of a computer program have not been clearly articulated.[332] At first glance, the object code version of a computer program appears to be a derivative work of the corresponding source code version of that computer program since compilation appears to translate human comprehensible source code into machine executable object code. The definition of "derivative work" in 17 U.S.C. § 101 reinforces this impression since it

328. In fact, it would be highly unusual to distribute such files with a commercial software program.

329. *See generally*, I. Trotter Hardy, Jr., *Six Copyright Theories for the Protection of Computer Object Programs*, 26 ARIZ. L. REV. 845 (1984). *See also*, Mathias Strasser, *A New Paradigm in Intellectual Property Law? The Case Against Open Source,* 2001 STAN. TECH. L. REV. 4, 31-4 (2001) (discussing the theoretical and doctrinal difficulties in justifying the extension of copyright protection to object code).

330. In fact, the various intermediate forms of a computer program do not appear to have been considered in a single reported case in the United States.

331. NIMMER & NIMMER, NIMMER, *supra* note 276, at §2.04 (stating that case law has held that a computer program in either source or object code is a form of "literary work" because it is "expressed in words, numbers or other verbal or numerical symbols or indicia"). Further, by definition, a computer program is "a set of statements or instructions to be used *directly or indirectly* inf a computer." 17 U.S.C. §101 (1999) (emphasis added). Because a computer cannot execute source code, the use of the word "indirectly" has been interpreted to mean that the source code form of a computer program is included in the definition of computer program and hence protected. Because a computer can execute object code, the use of the word "directly" has been interpreted to mean that the object code form of a computer program is also included within the definition of computer program and hence protected. The courts expressed some initial doubt. *See* Data Cash Sys., Inc. v. JS&A Group, Inc., 480 F.Supp. 1063 (N.D. Ill. 1979), *aff'd on other grounds*, 628 F.2d 1038 (7th Cir. 1980); Apple Computer, Inc. v. Franklin Computer Corp. (*Franklin Computer I*), 545 F.Supp. 812 (E.D. Pa. 1982). The jurisprudence now firmly supports copyright protection for computer programs in object code format. *See,* Apple Computer, Inc. v. Formula Int'l, Inc., 562 F.Supp. 775 (C.D. Cal. 1983) *aff'd*, 725 F.2d 521 (9th Cir. 1984); Apple Computer, Inc. v. Franklin Computer Corp. (*Franklin Computer II*), 714 F.2d 1240 (3rd Cir. 1983); Williams Elecs., Inc. v. Artic Int'l, Inc., 685 F.2d 870 (3rd Cir. 1982); Midway Mfg. Co. v. Strohon, 564 F.Supp. 741 (N.D. Ill. 1983).

332. Mathias Strasser, *supra* note 329, at 31-4.

includes translations as one of the listed examples of derivative works. Therefore, intuitively, the characterizing object code programs as derivative works of source code programs has great appeal.

However, some commentators have suggested that such an approach causes significant doctrinal difficulties.[333] It has been argued that the use of a compiler to create an object code program does not involve any originality from the developer of the corresponding source code program. In the United States, originality is a Constitutional pre-requisite for copyright protection; and a lack of originality will preclude copyright protection. It has been suggested that if there is any originality in the compilation process, the only possible source for that originality is the developer of the compiler.[334] This would lead to absurd and unworkable results because, theoretically, a developer of a source code program would not be able to distribute the object code version of that program without getting permission or license from the compiler developer.[335] Reference has also been made to United States Copyright Office practice, which does not allow separate registrations for the source and object code versions of a computer program because the Office maintains the view that there are no copyrightable differences between them.[336]

The position taken by the Copyright Office is not unreasonable. However, it is also possible to construct reasonable arguments for the opposite position. Copyright will subsist in a work if it is original, meaning that it has not been copied from another work and contains some minimal degree of creativity.[337] If an object code file is compared to its corresponding source code file there can be little doubt it has not been copied (at least in a literal sense) – one consists of mnemonics, mathematical operators and various other human readable content and the

333. *See Id. See also* Trotter Hardy, *supra* note 329, at 845, 850-52.

334. Mathias Strasser, *supra* note 329, at 33.

335. Strasser, *supra* note 329, at 33.

336. *See* United States Copyright Office, Compendium II: Copyright Office Practices, § 323.01(4), *available at* http://ipmall.info/hosted_resources/CopyrightCompendium/chapter_0300.asp. ("Since the object code version does not contain copyrightable differences; there is no basis for a separate copyright registration for the object code. The Office will communicate with the applicant suggesting a single registration for the computer program").

337. Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 345 (1991):

> [T]he *sine qua non* of copyright is originality. To qualify for copyright protection, a work must be original to the author. Original, as the term is used in copyright, means only that the work was independently created by the author (as opposed to copied from other works), and that it possesses at least some minimal degree of creativity. To be sure, the requisite level of creativity is extremely low; even a slight amount will suffice. The vast majority of works make the grade quite easily, as they possess some creative spark, 'no matter how crude, humble or obvious' it might be. Originality does not signify novelty; a work may be original even though it closely resembles other works so long as the similarity is fortuitous, not the result of copying." *Id.* (citations omitted).

other consists of binary processor instructions. Once the issue of copying has been dealt with, only the originality requirement remains. If two different, non-trivial source code programs are compiled, the resulting object code files will almost certainly be very different. The reason for this is that the source code input to a compiler has a huge impact on the object code produced by that compiler. Therefore, the creative effort of the author of the source code program should similarly have an enormous effect on the object code file produced by a compiler.

One of the arguments for the proposition that object code files are not derivative works of the corresponding source code files is that there is no subsequent author and, as a result, object code files cannot be original.[338] This argument assumes that none of the originality an author adds to a source code program is manifested in other forms of that program. However, this argument fails to consider the dual nature of software and in particular the dual nature or purpose of source code. Professor Randell Davis has observed that "[s]oftware is a 'machine' whose medium of construction happens to be text."[339] While the source code version of a program may at first glance appear to resemble other literary works, it differs in a number of important respects. The most fundamental of these differences is that the object code version of that source code program behaves. Programs exist in a textual format so they can be created and modified. One intended purpose for source code is to instruct and educate subsequent programmers who are going to maintain or enhance a computer program. These programmers must be able to read and understand a program so they can make the changes necessary to either repair or improve it. The second purpose for source code is to act as input to another program called a compiler,[340] to direct that program to construct an object code machine to perform the task for which the source code was initially created. The source code will have a profound effect on the object code machine actually constructed.

When a programmer writes a source code program, he or she will try to write that program in a clear and elegant manner so it can be easily understood by other programmers. Assuming a program is non-trivial, a programmer's efforts to write clear and elegant source code should infuse that code with the "creative spark" mandated by *Feist* to meet the originality requirement for copyrightability. However, programmers do not simply develop source code programs to be read by other programmers. Commercial computer programs are developed to operate on actual com-

338. Hardy, *supra* note 329, at 851.

339. Randell Davis, *Intellectual Property and Software: The Assumptions are Broken*, in WIPO WORLDWIDE SYMPOSIUM ON THE INTELLECTUAL PROPERTY ASPECTS OF ARTIFICIAL INTELLIGENCE 101, 110 (1991).

340. Source code programs may also contain instructions and input to be used by linkers and loaders.

puters. When a programmer writes a source code program, the programmer must consider the ultimate goal – the creation of a binary code program that will operate effectively on a computer. This goal will dictate many aspects of the overall design of the program. For example, a programmer may need to consider whether a particular piece of code can be multi-threaded, whether a particular instance of a data structure needs to be locked when it is updated, and whether there are real-time requirements that necessitate a maximum or minimum processing time for a particular operation. A programmer's efforts to write source code to address such considerations is once again likely to infuse the corresponding source code with the "creative spark" needed for originality. However, a programmer will not succeed unless his or her originality and creativity manifests itself in the end product – a functioning binary program. Accordingly, it does not seem reasonable that the law should limit the creative spark supplied by a programmer to the source code version of a program and extinguish that creative spark simply because it has been passed through a compiler. A compiler cannot create anything without the input and creativity supplied by the programmer who wrote the source code. A compiler is the mechanism by which some of the originality and creativity in the source code are conveyed to and ultimately manifested in an object code program. The fact that source code is used to convey this originality should not diminish the fact that this originality is actually manifested in the object code. Assuming that the source code provided to a compiler is original, it seems reasonable to conclude that the resulting object code is also original.

Another argument against object code being a derivative work of the corresponding source code program is based on the "purely mechanical means" restriction developed in connection with sound recordings.[341] This objection does not seem appropriate in the context of computer programs for a number of reasons. First, in the context of sound recordings, a mechanical means simply duplicates the work and there is no recasting or transformation of the work. Furthermore, a musical work always serves the same purpose no matter what the recording medium. As discussed above, computer programs in source and binary forms serve different purposes. Second, musical works are simply not particularly analogous to computer programs. Musical works have no functional purpose and are not created knowing that they will have to undergo another process to transform them to the form required for their main use. When a musician creates a musical work, his or her creativity is focused on one main goal - enjoyment of that work by an audience. When developing a computer program, a programmer focuses his or her creativity on comprehension, maintenance, and extension of the program by other pro-

341. Hardy, *supra* note 329, at 851-52.

grammers, and also on the operation of the binary version of the program on an actual computer.

Certain statements made in the final CONTU report have also been cited in support of the proposition that there is not enough originality in the object code version of a computer program for it to be a derivative work of the corresponding source code program.[342] In that report, in connection with a discussion about the use of a computer to create new works, the Commissioners stated that a musical work created by a computer could be copyrightable, as long as the would-be copyright owner "exercised sufficient control over the production of the work to be considered its author."[343] The CONTU report did not specify the level of control required for a creator to be considered an author in such circumstances. If one considers the creation of an object code program, it is evident that a compiler can do very little unless it is given detailed instructions - those instructions are given in the form of a source code program. A source code program acts as a series of instructions to a compiler for building an object code program. When viewed from this perspective, it is apparent that the developer of a source code program exerts a great deal of control over the creation of an object code program.[344] The CONTU Commissioners also observed that:

> Thus, it may be seen that although the quantum of originality needed to support a claim of authorship in a work is small, it must nevertheless be present. If a work created through application of computer technology meets this minimal test of originality, it is copyrightable. The eligibility of any work for protection by copyright depends not upon the device or devices used in its creation, but rather upon the presence of at least minimal human creative effort at the time the work is produced.

By the plain language of the report, the CONTU Commissioners were open to the creation of copyrightable works using computer technology. Furthermore, when the statements made in the CONTU report about computer-created works are properly considered it is evident that they actually support the proposition that an object code program has the requisite creativity for copyright protection. The only requirement in the CONTU report that appears to be even minimally troublesome is the requirement that the creative effort occur at the time the work is produced. However, it is unlikely the CONTU Commissioners intended to establish an arbitrary temporal requirement. From a copyright perspective, it should make no difference whether a creator interacts with a com-

342. *Id.* at 852.

343. *See* NAT'L COMM. ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS, FINAL REPORT 46 (1978), *available at* http://digital-law-online.info/CONTU/PDF/index.html.

344. *See contra* Hardy, *supra* note 329, at 850 (suggesting that a source code author typically has no control over the translation of the resulting object code).

puter-aided design tool in real-time or whether the creator interacts with that computer-aided design tool by creating a set of instructions, macros or data files that are then submitted to the tool. The situation with a source code program and a compiler is exactly analogous. When using a compiler, a developer is writing a set of instructions to be submitted to the compiler to create another work. Whether those instructions are passed to the compiler in a file or as real-time input should make no difference to the copyright status of the resulting object code program.

Assuming the creative effort does not need to occur at the instant a compiler is invoked, it would appear that a programmer exercises the creative effort necessary to produce copyrightable computer-generated object code. Indeed, the manner in which programs are tested and debugged further supports this proposition. Once a source code program has been written, it is compiled and linked to create an object code program. The object code program is then tested by operating it on a computer and observing its behavior. If the actual behavior does not match the desired behavior, then the program is debugged and modified until the desired behavior is actually observed in the object code program while it is operating on a computer. This iterative process provides empirical evidence that a significant portion of a programmer's creativity is directed towards and manifested in the object code program. The manner in which particularly difficult defects are resolved provides further evidence to support this proposition. When a defect is particularly difficult, a programmer will often examine the operation of an object code program in minute detail using a run-time debugger to set breakpoints and single step through object code instructions.

Accordingly, the creation of a commercial program is an iterative process, where a programmer alternately writes source code and tests object code to verify that his or her program works as intended and to ensure that his or her expression and creativity are properly manifested in the resulting object code program. If compilation was simply a mechanical translation, then this type of activity would not be necessary. The programmer would simply write his or her program so it was comprehensible to other programmers and the process would be complete. Since computer programs "behave," compilation and subsequent testing of the object code version of a program are critical steps in the creation of a final product. Previous derivative works analyses of object code programs have not taken into account the way in which computer programs are actually developed. As a result, the compilation process has been viewed as a formality that occurs once a computer program has been developed in source code. However, when the entire software development cycle is considered, it becomes apparent that compilation and testing are an integral part of the creative process for developing a computer program. Accordingly, when the entire development process is considered,

the argument that none of a computer programmer's creativity is embodied in an object code program is simply not persuasive.

Having said the foregoing, the law currently treats the object and source code forms (and presumably all other forms) of a computer program as different representations of the same work.[345] This view is probably incorrect since a technical analysis of compilation shows that at least some intermediate forms of a computer program can be derivative works of called libraries while other forms almost certainly are not. This suggests that the various forms of computer programs are not equivalent and hence cannot be simply different representations of the same work. If all of the forms of a computer program are simply different representations of the same work, then if one representation of a program is a derivative work of a linked library, an individual could presumably argue that distribution of any representation of the program should engage the GPL. The rationale being that the various representations of a work are not treated as separate works and if one of those representations engages the viral provisions of the GPL, then presumably, all of the representations of that work should engage the viral provisions of the GPL. However, this proposition seems overly broad since the various forms of a computer program clearly have different substantive content and are not really equivalent representations of the same work. A more appropriate approach would be to deal with the different forms of a computer program as separate works with their exact treatment and relationship to each other to be determined by their specific content. Accordingly, a GPL analysis should focus only on those forms of a computer program that are distributed – most often the source code and object code. Therefore, the fact that an intermediate form of an object code program is almost certainly a derivative work of a linked GPL-licensed library should not make any difference. The test to be applied to any object or source code distribution should be whether that form of the program contains a substantially similar amount of copyrightable material thereby making it a derivative or infringing work of a linked library.

If it is subsequently determined that the object code form of a computer program is a derivative work of the source code or any other intermediate form of a program, this change should not affect the analysis proposed above. According to Professor Jay Dratler, whether a subsequent work is a derivative work of an earlier work will always depend on whether there has been an inclusion of a substantially similar amount of copyrightable material from the earlier work.[346] Accordingly, if a third

345. *See* Strasser, *supra* note 329. *See also* Compendium II: Copyright Office Practices, *supra* note 336.

346. JAY DRATLER, INTELLECTUAL PROPERTY LAW: COMMERCIAL, CREATIVE, AND INDUSTRIAL PROPERTY, §6.01[2] (1998).

work is a derivative work of a second work and the second work is a derivative work of a first work, the third work will only be a derivative work of the first work if it is substantially similar to the first work. Professor Dratler further observed that the number of cases in which such a third work will not be a derivative work of the first work is likely to be low.[347] Therefore, assuming a final object code version of a computer program is a derivative work of the source code file or the intermediate file, and the source code file or intermediate file is a derivative work of a dynamically linked GPL-licensed library, the question of whether the object code program is also a derivative work of the GPL-licensed library will ultimately be resolved by determining whether the object code program contains enough copyrightable material from that GPL-licensed library to make it substantially similar. This is the same test as in the case where the object code program is a separate work from all the other versions of the computer program.

Thus, the question of whether the dynamic linking of a commercial program to a GPL-licensed library makes that program a derivative work comes down to an analysis of whether the resulting object code program is substantially similar to the linked library. As discussed in Section II.C, a dynamically linked object program will contain the various symbol names referenced from any linked GPL-licensed libraries. The object code in the dynamically linked program may also contain non-literal elements copied from any linked libraries. Critically, however, a dynamically linked executable object program will not contain any object code from the libraries to which it has been linked. Therefore, the analysis of whether an object code program is a derivative work of a GPL-licensed library to which it has been linked ultimately becomes a fact-dependent analysis of whether the symbol names and any non-literal elements that may have been copied are sufficient to meet the substantial similarity test. This is a somewhat unsatisfying result, since it means, absent other limiting doctrines, every use of a GPL-licensed library must be analyzed in detail and, specifically, the nature of the interaction with that library must be very well understood before a determination can be made about the potential viral effects of GPL-licensed software use.[348]From the perspective of a commercial software vendor the answer is even more unsatisfying since the substantial similarity test does not provide a bright-line answer and, accordingly, there will almost always be some level of uncertainty about each use of GPL-licensed software. Therefore, each time a commercial software vendor decides to dynami-

347. *Id.* at §6.01[2] 6-12.2-12.2.3. As will be discussed later, for computer programs, and particularly for programs that are dynamically linked or that communicate through inter-process communication, this type of situation may not necessarily be that uncommon.

348. 17 U.S.C. §102(b) (1990). Later it will be argued that the limiting provisions of 17 U.S.C. §102(b) should apply to dynamic linking.

cally link a commercial program to a GPL-licensed library, that vendor will be taking a non-zero risk of compromising the value of the commercial program. However, as will be discussed later, there are some situations where the GPL-related risk may be significantly less than the typical case.

2. *Copyright Protection for Technical Interfaces*

The various names, parameters and data structures used to call functions and procedures in a dynamically linkable GPL-licensed library constitute a technical interface. At a high level, a technical interface is a set of words and rules that must be used and observed by someone who wants to interact with a given program.[349] Technical interfaces perform a similar function to the desktop and file folder interface paradigms used in various popular operating systems. However, instead of facilitating interactions between humans and computers, technical interfaces facilitate interactions between computer programs/processes and other computer programs/processes. Much of the early jurisprudence on interface-related issues was developed in the context of computer-human interfaces.[350] Even now, the number and significance of cases dealing with computer-human interfaces significantly outweighs those dealing with technical interfaces. The irony is that there are significantly more computer-to-computer and program-to-program interfaces than there are computer-human interfaces. However, the number of cases involving technical interfaces seems to be increasing. Additionally, a number of important principles developed in the context of graphical user interfaces are also highly relevant for technical interfaces.

---

349. Zimmerman, *supra* note 306, at 10.

350. *See generally* Lotus Dev. Corp. v. Borland Int'l., Inc. (*Lotus V*), 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer*), 35 F.3d 1435 (9th Cir. 1994); Autoskill, Inc. v. Nat'l Educ. Support Sys., Inc., 994 F.2d 1476 (10th Cir. 1993); Brown Bag Software v. Symantec Corp., 960 F.2d 1465 (9th Cir. 1992); Ashton-Tate Corp. v. Ross, 916 F.2d 516 (9th Cir. 1990); Lotus Dev. Corp. v. Borland Int'l Inc. (*Lotus IV*), 831 F.Supp. 223 (D. Mass. 1993); Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus III*), 831 F.Supp. 202 (D. Mass. 1993); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer VI*), 821 F.Supp. 616 (N.D. Cal. 1993); Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus II*), 799 F.Supp. 203 (D. Mass. 1992); Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus I*), 788 F.Supp. 78 (D. Mass. 1992); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer V*), 799 F.Supp. 1006 (N.D. Cal. 1992); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer IV*), 779 F.Supp. 133 (N.D. Cal. 1991); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer III*), 759 F.Supp. 1444 (N.D. Cal. 1991); Lotus Dev. Corp. v. Paperback Software Int'l, 740 F.Supp. 37 (D. Mass. 1990); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer II*), 717 F.Supp. 1428 (N.D. Cal. 1989); Apple Computer, Inc. v. Microsoft Corp. (*Apple Computer I*), 709 F.Supp. 925 (N.D. Cal. 1989); Mfr. Tech., Inc. v. Cams. Inc., 706 F.Supp. 984 (D. Conn. 1989); Digital Commc'n Assocs., Inc., v. Softklone Distrib. Corp., 659 F.Supp. 449 (N.D. Ga. 1987); Broderbund Software, Inc. v. Unison World, Inc., 648 F.Supp. 1127 (N.D. Cal. 1986).

3. *Cases Relevant to Copyright Protection for Technical Interfaces*

The current case law relating to technical interfaces is mixed.[351] In the next section, the most significant cases relating to copyright protection for technical interfaces will be analyzed and critiqued. The principles derived from these cases will then be applied to dynamic linking of GPL-licensed libraries.

i. Lotus

Perhaps the most significant case to consider when attempting to determine the scope of protection for technical interfaces is a case that does not appear to deal with technical interfaces at all. This is the famous case *Lotus Development Corp. v. Borland International, Inc.* that dealt with the copyrightability of the Lotus 1-2-3 command names and command menu hierarchy.[352]

Lotus 1-2-3 is a graphical spreadsheet program that allows users to perform various accounting functions on a computer.[353] Users are able to control the program and manipulate accounting information using a large number of mnemonic menu commands such as "Copy", "Print" and "Quit."[354] The Lotus program also allows users to write programs called "macros" consisting of Lotus 1-2-3 commands.[355] Lotus 1-2-3 can read macro files and execute the commands contained within the macro thus saving the user from having to type them.[356] Lotus 1-2-3 was very commercially successful and the menu command hierarchy was characterized as the *de facto* standard for electronic spreadsheet commands.[357]

A number of years after the release of Lotus 1-2-3, Borland released competing electronic spreadsheet programs called Quattro and Quattro Pro.[358] Evidence in the case showed that Borland had done nearly three years of development work on Quattro and Quattro Pro, and Borland's objective had been to create programs that were far superior to those currently on the market.[359] However, each of the Borland products contained "virtually identical" copies of the entire Lotus 1-2-3 command hi-

351. *See*, Zimmerman, *supra* note 306, at 10.

352. Lotus Dev. Corp. v. Borland Int'l., Inc., (*Lotus V*), 49 F.3d 807, 809 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam).

353. *Id*.

354. *Id.* The version of Lotus 1-2-3 considered by the court contained 469 commands arranged into more than 50 menus and submenus. *Id.*

355. *Id*.

356. *Id.* The process of reading and executing a macro is also known as "invoking" that macro.

357. *Lotus V*, 49 F.3d at 821 (Boudin J., concurring).

358. *Lotus V*, 49 F.3d at 810.

359. *Id.*

erarchy.[360] Borland included the Lotus command hierarchy in its products to make them compatible with Lotus 1-2-3, so users who were already familiar with Lotus 1-2-3 could more easily transition to the Borland products.[361] Additionally, the Quattro and Quattro Pro compatibility features were designed to allow Lotus users to continue to use macros they had written for Lotus 1-2-3. Compatibly was achieved in two ways.[362] The first way was through the provision of an alternative user interface, known as the "Lotus Emulation Interface."[363] While the Borland products had their own native user interfaces, that were different from the Lotus 1-2-3 user interface, the Borland products also allowed users to switch to the Lotus Emulation Interface, which supported all of the Lotus 1-2-3 commands.[364] Except for minor differences, when running in Lotus emulation mode, Borland's products looked quite similar to the Lotus 1-2-3 graphical user interface.[365] The second way in which compatibility was provided was through the "Key Reader" feature in the Quattro Pro product.[366] The Key Reader allowed the Quattro Pro product to read and execute Lotus 1-2-3 macros.[367] Borland developed these compatibility features without copying any of the underlying Lotus source code – only the words and structure of the Lotus command hierarchy was copied.[368] However, this copying was virtually complete.[369]

At trial, the district court found in favor of Lotus, and held that Borland's copying of the Lotus 1-2-3 menu commands and menu command hierarchy was an infringement of Lotus' copyright in that product.[370] The district court stated that "the 'menu commands' and 'menu structure' contain expressive aspects separable from the functions of the 'menu commands' and 'menu structure.'"[371] However, the First Circuit Court of Appeals did not agree with the district courts' approach or conclusions, and overruled that decision. After dealing with a number of preliminary matters,[372] the First Circuit focused its analysis on whether

360. *Id.*
361. *Id.*
362. *Id.*
363. *Id.*
364. *Lotus V*, 49 F.3d at 810.
365. *Id.* at 810.
366. *Id.* at 811.
367. *Id.* 811.
368. *Id.* at 810.
369. *Id.* at 810.
370. Lotus Dev. Corp. v. Borland Int'l, Inc., (*Lotus II*), 799 F.Supp. 203, 205 (D. Mass. 1992).
371. *Id.* at 223.
372. *Lotus V*, 48 F.3d at 814 (stating that the case was not identical to Baker v. Selden, 101 U.S. 99 (1879)). *Id.* at 814-15 (stating that Computer Assoc. Int'l, Inc. v. Altai, Inc. 982 F.2d 693 (2nd Cir. 1992) was not particularly helpful as an analytic framework because that case dealt with non-literal copying, while Lotus dealt with deliberate literal copying).

the Lotus command hierarchy was a method of operation.

Section 102(b) of the Copyright Act provides that "[i]n no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated or embodied in such work."[373] The First Circuit interpreted the term "method of operation" as used in §102(b) to refer to "the means by which a person operates something, whether it be a car, a food processor, or a computer."[374] The court also observed that the Lotus command hierarchy provided the means by which users of Lotus 1-2-3 control and operate the program.[375] Further, the First Circuit observed that users would not be able to access and control or make use of the Lotus 1-2-3 functional capabilities without the menu command hierarchy.[376] Based on its understanding of the meaning of the term "method of operation," and based on its observations about the Lotus menu command hierarchy, the First Circuit concluded that the Lotus 1-2-3 menu commands and menu command hierarchy were not entitled to copyright protection.[377] Accordingly, the court found that Borland's use of the Lotus 1-2-3 menu commands and menu command hierarchy was not copyright infringement.[378]

The First Circuit also explained its reasons for rejecting the district court's approach. As will be discussed later, the reasons the First Circuit gave for rejecting the district court approach are very applicable to dynamic linking. The First Circuit stated that it accepted the district court's premise that certain "expressive choices" had been made in choosing and arranging the Lotus 1-2-3 commands.[379] The court also said it did not believe that methods of operation were limited to abstractions.[380] In particular, the court found that "[i]f specific words are essential to operating something, then they are part of a 'method of operation' and, as such, are unprotectable."[381] The First Circuit further explained that:

> The fact that Lotus developers could have designed the Lotus menu command hierarchy differently is immaterial to the question of whether it is a "method of operation." In other words, our initial inquiry is not whether the Lotus menu command hierarchy incorporates any expression. Rather, our initial inquiry is whether the Lotus menu command hierarchy is a "method of operation." Concluding, as we do, that users

373. 17 U.S.C. §102(b) (2010).
374. *Lotus V*, 48 F.3d at 815.
375. *Id*.
376. *Id.*
377. *Id.* at 818.
378. *Id.* at 819.
379. *Id.* at 816.
380. *Lotus V*, 48 F.3d at 816.
381. *Id.*

> operate Lotus 1-2-3 by using the Lotus menu command hierarchy, and that the entire Lotus menu command hierarchy is essential to operating Lotus 1-2-3, we do not inquire further whether that method of operation could have been designed differently. The "expressive" choices of what to name the command terms and how to arrange them do not magically change the uncopyrightable menu command hierarchy into copyrightable subject matter.[382]

The court observed that Lotus wrote its menu command hierarchy "so that people could learn and use it," and accordingly, it fell within the prohibition on copyright protection established in *Baker v. Selden* and codified in §102(b).[383] It should be noted that the First Circuit's holding was limited to the menu commands and menu command hierarchy. The decision did not cover other parts of the Lotus 1-2-3 interface, such as the various screen displays. The reason for this limitation is that these aspects of the Lotus 1-2-3 interface were not at issue by the time the case reached the First Circuit. However, by extending the court's reasoning, it seems reasonable to conclude that these aspects of the Lotus 1-2-3 interface would not have been denied copyright protection because they would not have been essential to the operation of Lotus 1-2-3.

Since *Lotus*, a number of other circuit courts have decided to not follow the First Circuit's approach.[384] These decisions seem to arise from the perceived harshness of an absolute ban on copyright protection for the expressive elements of a method of operation. The First Circuit's reasoning is certainly justified on a simple reading of the language of §102(b). However, questions have arisen about whether the First Circuit

---

382. *Id.* at 816 (footnote omitted) (emphasis added).

383. *Id.* at 817.

384. Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1372 (10th Cir. 1997) ("[W]e decline to adopt the *Lotus* court's approach to section 102(b), and continue to adhere to our abstraction-filtration-comparison approach"). Mitek Holdings, Inc. v. Arce Eng'g Co., Inc., 89 F.3d 1548, 1557 (11th Cir. 1996) ("Unlike the Lotus court, we need not decide today whether a main menu and submenu command tree structure is uncopyrightable *as a matter of law*"). Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1547 (11th Cir. 1996). Without referring to *Lotus V* on this point, the Eleventh Circuit said:

> This circuit has yet to address whether interface specifications, *as a matter of law*, are not entitled to copyright protection. In its briefs, PAC argues that Bateman's interface commands are legally uncopyrightable and, thus, PAC was not obligated to avoid copying them by rewriting its application program. . . . To the extent that [PAC] was making the former argument, we reject it. It is an incorrect statement of the law that interface specifications are not copyrightable as a matter of law. (footnotes omitted).

In Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1547, n.31 (11th Cir. 1996), the Eleventh Circuit further added:

> We need not decide whether PAC is correct in its assertion that, given the particular facts of this case, it was not obliged to rewrite its application program to avoid copying Bateman's interface specifications. PAC, however, is incorrect in arguing that this rewriting was not required because Bateman's interface specifications are not entitled to copyright protection *as a matter of law*.

approach implements the legislative intent of §102(b). An examination of *Baker v. Selden*, which enunciated many of the principles subsequently codified in §102(b), shows that the United States Supreme Court did not proscribe copyright protection of all expression within a method of operation, but instead, proscribed copyright protection for expression that is necessary for the use of a method of operation.[385] Accordingly, the First Circuit's finding that the menu commands and menu command hierarchy are not protected under copyright appears to give precise effect to this narrower interpretation of §102(b) and the legislative intent related to that section. In particular, the court justified the *Lotus* result:

> [O]ur inquiry is not whether the Lotus menu command hierarchy incorporates any expression. Rather, our initial inquiry is whether the Lotus menu command hierarchy is a "method of operation." Concluding, as we do, that users operate Lotus 1-2-3 by using the Lotus menu command hierarchy, *and that the entire Lotus menu command hierarchy is essential to operating Lotus 1-2-3*, we do not inquire further whether that method of operation could have been designed differently. The "expressive" choices of what to name the command terms and how to arrange them do not magically change the uncopyrightable menu command hierarchy into copyrightable subject matter.[386]

The *Lotus* holding will not generally extend to most arrangements of icons or other elements on a screen display because those arrangements are generally not "essential to operating" a given product. Screen displays will usually contain expressive choices that should be protected even under a *Lotus*-like analysis because those displays are not essential to the operation of the product.[387]

---

385. In Baker v. Selden, 101 U.S. 99, 103 (1879), the Court said:

> The copyright of a work on mathematical science cannot give to the author an *exclusive right* to the methods of operation which he propounds, or to the diagrams which he employs to explain them, *so as to prevent an engineer from using them whenever occasion requires.* The very object of publishing a book on science or the useful arts is to communicate to the world the useful knowledge which it contains. But this object would be frustrated if the knowledge could not be used without incurring the guilt of piracy of the book. *And where the art it teaches cannot be used without employing the methods and diagrams used to illustrate the book, or such as are similar to them, such methods and diagrams are to be considered as necessary incidents to the art, and given therewith to the public; not given for the purpose of publication in other works explanatory of the art, but for the purpose of practical application.*
>
> Of course, these observations are not intended to apply to ornamental designs, or pictorial illustrations addressed to the taste. (emphasis added).

386. Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*) 49 F.3d 807, 816 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam) (footnote omitted) (emphasis added).

387. This approach will also produce reasonable results when applied to other contexts such as, operating systems that are based on desktop metaphors. The visual screen displays of such operating systems, while part of the method for operating that program, should not be denied copyright protection simply because they are used in operating that

The First Circuit went on to consider the Lotus 1-2-3 macros, and similarly concluded that those macros were an unprotectable method of operation. This part of the *Lotus* analysis is particularly applicable to dynamic linking since Lotus 1-2-3 macros are created in much the same way as computer programs. Additionally, when macros are used, they rely on a technical interface between the macro and the application. Accordingly, the First Circuit's finding that the Lotus 1-2-3 macros are a method of operation is significant because dynamic linking relies on a technical interface between a calling program and a linked library. In its analysis of the macro capabilities in the Lotus and Borland products, the First Circuit said:

> We also note that in most contexts, there is no need to "build" upon other people's expression, for the ideas conveyed by that expression can be conveyed by someone else without copying the first author's expression. In the context of methods of operation, however, "building" requires the use of the precise method of operation already employed; otherwise building would require dismantling, too. Original developers are not the only people entitled to build on methods of operation they create; anyone can.[388]

This passage articulates the reason why the particular words and formats of a GPL-licensed dynamically linkable library API are unlikely to be granted copyright protection under a *Lotus*-like approach. If those mnemonics and formats are not available for unencumbered use, then the functionality in the underlying library cannot be operated. Indeed, the reason for creating an API is to allow other programmers to access and use the underlying functionality. It is therefore difficult to envision any strong arguments for the proposition that a GPL-licensed library's API is not a method of operation for the underlying library.

Therefore, the rationale for finding that the Lotus 1-2-3 macros are a method of operation also seems to be applicable to dynamic linking, thus making *Lotus* very relevant in determining the copyright treatment of dynamic linking. However, while macros are very program-like and

---

program. To the extent expressive choices have been made, and to the extent that expression is not filtered for other reasons, these choices should be granted protection because their use is not required to use the methods of operation for a desktop-metaphor based operating system. However, to the extent the same operating system provides services to application programs, or otherwise makes application programming interfaces, or, other technical interfaces available so that capabilities of the operating system can be used by other programs, those application programming interfaces, or other technical interfaces, should be denied copyright protection under the Lotus approach because the specific words, data structures, and formats used in these methods of operation are essential to their use. If such words, data structures and formatting were protected, the effect would be to deny access to and use of those methods of operation- – conflicting with the explicit provisions of §102(b) and the holding in Baker v. Selden 101 U.S. 99 (1879).

388. *Lotus V*, 49 F.3d at 818 (footnotes omitted).

while there is a technical interface between a macro and the application program that executes that macro, the analogy is not identical. The First Circuit's specific reasons used to justify its finding that the Lotus 1-2-3 macros were a method of operation are not as clearly stated as those for the menu commands and menu command hierarchy. Much of the rationale seems to be based on compatibility requirements.[389] However, computer program compatibility does not justify a denial of copyright protection.[390]

The need for a subsequent program to establish compatibility with a pre-existing program is not a reason for denying copyright. In analyzing the Lotus 1-2-3 macros, the First Circuit should have applied the same two-step analysis it used for the menu commands and hierarchy. Had that approach been used, the court would have assessed whether the subject matter at issue is the means by which something is operated. For the macro command facility, this requirement appears to have been met since the macro reader provides an alternative means by which commands can be submitted to Lotus 1-2-3 and processed. Once the macro reader facility is determined to be a method of operation, the expression necessary to use it will not be protected. In this case, this material would probably have largely overlapped with the material in the macro commands and macro command hierarchy.

The First Circuit completed its analysis by observing that if someone were to write a macro based on the Lotus 1-2-3 menu commands, the district court's ruling would prevent that person from using the macro to perform the same operations with another program.[391] The court further observed that if such a person wanted to use the macro with another program, that person would have to re-write the macro for the other pro-

389. In particular, the court in *Lotus V* observed:

> That the Lotus menu command hierarchy is a "method of operation" becomes clearer when one considers program compatibility. Under Lotus's theory, if a user uses several different programs, he or she must learn how to perform the same operation in a different way for each program used. For example, if the user wanted the computer to print material, then the user would have to learn not just one method of operating the computer such that it prints, but many different methods. We find this absurd. *Lotus V*, 49 F.3d at 817-18.

390. The purpose of U.S. copyright law is to promote the progress of science and the useful arts. The grant of copyright protection is a means to achieve that end by providing an incentive for authors to create works. This grant, however, is not simply meant to reward authors. Compatibility tends to promote progress of sciences and the useful arts; therefore it is a desirable result. Therefore, compatibility, along with a number of other factors that promote the sciences and the useful arts, are considerations that will tend to influence how the Copyright Act is construed and applied. However, compatibility is not a statutorily mandated reason for denying copyright protection; although it may be a consideration in a fair use argument.

391. *Lotus V*, 49 F.3d at 818.

gram.[392] The First Circuit stated that forcing a user to cause a computer to perform the same operation in a different way ignores the fact that methods of operation are not copyrightable.[393] This reasoning is problematic in two ways. First, it is broader in scope than the rationale used earlier in the decision, which only denied protection to those aspects of a method of operation necessary for its use. Second, user convenience and compatibility are consequences of a finding that the Lotus 1-2-3 menu commands and menu command hierarchy are a method of operation, not a reason for finding that the Lotus 1-2-3 menu commands and menu command hierarchy are a method of operation. Finally, the First Circuit concluded that "[a]s the Lotus menu command hierarchy serves as the basis for Lotus 1-2-3 macros, the Lotus menu command hierarchy is a 'method of operation.'"[394] This statement is somewhat puzzling, and it appears that the court meant to say that since the Lotus menu command hierarchy serves as the basis for the Lotus 1-2-3 macros, the Lotus 1-2-3 macros are a method of operation. In any event, the overall holding of the case makes it clear that Borland was not liable for copying the menu commands, the menu command hierarchy, or the macro reading feature. Thus, this statement cannot reasonably mean anything except that the macro facility is a method of operation. Therefore, there are actually two methods of operation for the Lotus 1-2-3 product: one consisting of the menu commands and menu command hierarchy, which are typed or selected through a graphical user interface, and the other consisting of the macro reading facility and macro commands, which are typed into a computer file and then read and executed by using the macro reading feature.

Even though the settings are not identical, it appears that the First Circuit's rationale in *Lotus* can be reasonably applied to dynamic linking. Unlike the situation in *Lotus*, an end user will not engage a dynamically linkable library through a user interface. Instead, a dynamically linkable library will be called from an executing program through the library's API. According to *Lotus*, the term "method of operation" refers to "the means by which a person operates something, whether it be a car, a food processor, or a computer."[395] At first glance, it appears that the reference to "person" might require some kind of direct human participation. However, the requirement for direct human participation was not an impediment to the finding that the Lotus 1-2-3 macro reading facility was a method of operation.[396] Later in the *Lotus* opinion, the First Cir-

392. *Id.*
393. *Id.*
394. *Id.*
395. *Id.*
396. It could also be argued that the user who selects the macro reading facility supplies the necessary human activity. Assuming there is a requirement for human activity to qual-

cuit stated. "[i]f specific words are essential to operating something, then they are part of a 'method of operation' and, as such are unprotectable. This is so whether they must be highlighted, typed in, or even spoken, as computer programs no doubt will soon be controlled by spoken words."[397] While the First Circuit gives examples that all involve human participation, the test itself does not contain any such limitation, and it is broad enough to include one computer program invoking another computer program. Additionally, other circuit courts have shown no inclination to restrict methods of operation to circumstances involving direct human participation.[398] Leaving aside the issue of direct human participation, there does not appear to be anything else in the First Circuit's test that would prevent a GPL-licensed API from qualifying as a method of operation. In particular, the only way to invoke a dynamically linkable library is by embedding its API calls in an application program that subsequently links to that GPL-licensed library in an operating computer. Furthermore, the specific API names, parameter lists and data structures are essential to the dynamic linking process. If the precise semantics specified by the API are not followed, a linking application program will not compile or it will not function properly. Therefore, based on these characteristics, it would appear that invocation of a GPL-licensed dynamically linkable library through its API should qualify as a method of operation.

ii. Bateman

About a year after the decision in *Lotus v. Borland*, another circuit court had an opportunity to consider a fact scenario that is quite similar to dynamic linking and inter-process communication. In *Bateman v. Mnemonics, Inc*., the Eleventh Circuit Court of Appeals considered the

---

ify as a method of operation, it is also possible that invocation of a program could potentially supply the necessary human activity. Additionally, the actions of a programmer in adding an API call to a program could be sufficient to meet a requirement for human activity. These few examples demonstrate that the imposition of a human activity requirement does not result in an easy or principled way for identifying methods of operation, and subsequent cases have not tried to impose such a requirement. Accordingly, it seems unlikely that the First Circuit Court of Appeals intended human participation to be a prerequisite. *See*, Gates Rubber Co. v. Bando Chemical Indus., Ltd., 9 F.3d 823, 836 n.13 (10th Cir. 1993) (characterizing procedures, processes, systems and methods of operation as "methods for achieving a particular result").

397. *Lotus V*, 49 F.3d at 816.

398. *See e.g.*, Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1368 (10th Cir. 1997) (describing the district court's finding that Mitel's command codes comprised the method by which a long distance carrier matches the call controller's functions, the carrier's technical demands, and the telephone customers' choices, and then concluding that although an element of a work may be characterized as a method of operation, that element may nonetheless contain expression that is eligible for copyright protection).

scope of copyright protection for various system calls required to communicate with a single board computer operating system ("SBCOS").[399] The defendant had been a licensee of the plaintiff's SBCOS, and had written a number of application programs that interacted with that operating system.[400] Eventually, the business relationship between the plaintiff and the defendant deteriorated and the license agreement for the SBCOS was terminated.[401] After termination of the license, the defendant decided to develop its own operating system that would interoperate with the application programs it had already developed for the SBCOS.[402] To develop its own operation system, the defendant dissembled the SBCOS to identify those parts of the operating system necessary for interoperation with the existing application programs.[403]

The primary issue on appeal involved certain jury instructions given by the district court.[404] Specifically, the defendant contended that the district court had erred when it instructed the jury to filter out only nonliteral similarities when performing the second step of the abstraction-filtration-comparison test.[405] The defendant also claimed that the district court erred when it did not instruct the jury on the legal consequences of finding that certain literal instances of copying by the defendant were dictated by compatibility and interoperability requirements.[406] The Eleventh Circuit concluded that the jury had not been properly instructed on either issue, and remanded the case back to the district court for re-trial.[407]

In considering the second alleged error, the Eleventh Circuit made certain comments that are potentially relevant to the scope of protection for technical interfaces. In particular, the court examined whether to deny interface specifications copyright protection as a matter of law.[408] In this case, the reference to interface specifications is really a reference to the API that the SBCOS made available for use by application programs intended to run on that operating system to use. The Eleventh Circuit concluded that it is incorrect to say that interface specifications are not copyrightable as a matter of law.[409] In reaching this conclusion, the court did not consider the First Circuit's decision in *Lotus v. Borland*, despite the fact that application of the First Circuit's reasoning to this

399. Bateman v. Mnemonics, Inc., 79 F.3d 1532 (11th Cir. 1996).
400. *Id.* at 1537.
401. *Id.* at 1539.
402. *Id.*
403. *Id.*
404. *Id.* at 1540.
405. *Bateman*, 79 F.3d at 1543.
406. *Id.* at 1546.
407. *Id.* at 1550.
408. *Id*. at 1547.
409. *Id.*

case could have led to a different conclusion reached by the Eleventh Circuit.

The technical interface in *Batemen* consisted of various system calls made available by the SBCOS for use by application programs requesting services from the SBCOS.[410] These system calls were the means by which application programs used functionality provided by the SBCOS.[411] Furthermore, the specific words of those system calls, such as the names of those calls and the parameter lists and data structures used by those calls, were required to allow application programs to use the capabilities provided by the SBCOS. Therefore, pursuant to the criteria established in *Lotus*, these interface specification, or system calls, should qualify as a method of operation. According to *Lotus*, once a court finds that certain subject matter is a method of operation, it no longer matters whether a developer made expressive choices because, under §102(b), as a matter of law, the expression necessary for the use of that method of operation cannot be protected.[412] However, even under *Lotus*, the statement that an interface specification in its entirety is not copyrightable as a matter of law is not true since there may be aspects of an interface specification that are not necessary for its use. Generally, however, there will be significantly less non-essential material in a technical interface than in a computer-human interface. Therefore, while the Eleventh Circuit in *Bateman* was correct in stating that not all interface specifications are denied copyright protection as a matter of law, the Eleventh Circuit appears to have failed to consider whether this particular specification might be one for which copyright protection might be

410. *Id.* at 1537.

411. The description of operating systems and application programs in Computer Assoc. Int'l, Inc. v. Altai, Inc., 775 F.Supp. 544, 549-50 (E.D.N.Y. 1991) illustrates this relationship:

> Operating systems are the programs that manage the resources of the computer and allocate those resources to other programs that need them. For example, operating system software might perform, among others, these functions:
> - channeling information entered at a keyboard to the proper application program;
> - sending information from an application program to a display screen;
> - providing blocks of memory to an application program that requires them; and
> - allocating processing time among several application programs running on the computer at the same time.
>
> Operating system software interacts with whatever other programs are being used or "executed" by the computer, providing computer resources such as processors, memory, disk space, printers, tape drives, etc. for the other programs that need them through what are often referred to as "system calls". For this interaction to occur properly, the other programs must be compatible with the operating system software in use on the computer, i.e., they must be able to exchange information precisely and accurately with the operating system to interact with those computer resources. Computer Assoc. Int'l, Inc. v. Altai, Inc., 775 F. Supp. 544, (E.D.N.Y. 1991) (emphasis added).

412. Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*), 49 F.3d 807, 816-19 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam).

denied as a matter of law. Although the *Bateman* decision did refer to *Lotus* on other matters, the Eleventh Circuit either did not notice that its treatment of methods of operation conflicted with *Lotus*, or it chose not to comment on that conflict.[413] In any event, the court's failure to consider the SBCOS interface more closely, or to at least provide a rationale for not considering the interface more closely, makes it difficult to assess the merits of the *Bateman* decision.

iii. Mitel

A little over two years after the First Circuit Court of Appeals decided *Lotus v. Borland*, the Tenth Circuit Court of Appeals had an opportunity to consider the scope of copyright protection for technical interfaces.[414] In *Mitel*, *Inc. v. Iqtel, Inc*., the Tenth Circuit concluded that copyright protection should not be extended to the technical interface at issue in that case.[415] However, the Tenth Circuit explicitly rejected the approach taken by the First Circuit in *Lotus v. Borland* and established a much different scope of protection for technical interfaces.[416]

Mitel manufactured and sold a product known as a call controller.[417] A call controller is a specialized piece of computer hardware that automates the selection of long distance carriers and also automates the selection of various optional telephone features such as speed dial.[418] The Mitel call controller was activated and manipulated using an instruction set consisting of over sixty numeric command codes.[419] Each command code consisted of a four-digit sequence selected from those digits generally available on a telephone keypad, such as zero through nine, * and #.[420] The various digits within a command were used to specify information, such as the function to be executed and the telephone line on which that function was to be executed. Other digits were used to specify the value of certain parameters that might be applicable to a particular command.[421] For example, in the case of some commands, a "4" might represent a value of 1200 baud; in other cases a "4" might represent a

413. Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1542-45 (11th Cir. 1996).

414. Zimmerman, *supra* note 306, at 12. Although the Mitel interface bears some resemblance to a user interface when being installed or updated by technicians, in operation it functions as a technical interface. Once installed, the call controller is automatically invoked by a telephone company's networks and systems with no human interaction or intervention.

415. Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1375 (10th Cir. 1997).

416. *Id.* at 1371-73.

417. *Id.* at 1368.

418. *Id.*

419. *Id.*

420. *Id.*

421. *Mitel*, 124 F.3d at 1369.

value of forty seconds.[422]

Iqtel was a subsequent entrant into the market for call controllers.[423] Iqtel designed its own call controller with its own set of command codes.[424] While the Iqtel and Mitel call controllers performed many of the same functions, many parts of the respective command sets were different; however, for the parts of the command sets that represented parameter values, Iqtel selected all of the same ranges as Mitel.[425] At the time Iqtel entered the call controller market, the Mitel product dominated the market.[426] Because of this dominance, Iqtel did not believe their call controller would be successful unless it was compatible with the Mitel product.[427] Specifically, Iqtel did not believe that technicians who installed call controllers would be willing to learn Iqtel's new command set in addition to the Mitel command set.[428] Accordingly, Iqtel designed their product to accept Mitel commands, which were then translated into their Iqtel equivalents. The Iqtel controller then executed these equivalent commands.[429] The Iqtel product proved to be highly competitive with the Mitel product and, as a result, Mitel brought an action against Iqtel for copyright infringement of the Mitel command codes.[430] Iqtel did not dispute that it had copied Mitel's command codes.[431]

The district court found for Iqtel, holding that the Mitel command codes were unprotectable because the codes were: (i) a method of operation under 17 U.S.C. §102(b), (ii) unoriginal under 17 U.S.C. §102(a), and (iii) dictated by external factors, and hence unprotectable under the scènes à faire doctrine.[432] The Tenth Circuit upheld the district court

422. *Id.*
423. *Id.*
424. *Id*.
425. *Id.*
426. *Id.*
427. *Mitel*, 124 F.3d at 1369.
428. *Id*.
429. *Id*.
430. *Id.* at 1370.
431. *Id.*
432. Mitel, Inc. v. Iqtel, Inc., 896 F.Supp. 1050, 1055 (D.Colo. 1995) (holding:

> The command codes are simply a procedure, process, system, and method of operation by which the customer can match the call controller functions to the long-distance carriers' technical needs and the end user's choices. Without the command codes the function would not occur and the result would not be achieved. Consequently, I conclude and hold that the command codes are not protected components of Mitel's copyrighted material.
>
> I arrive at the same conclusion applying the *Gates Rubber* abstraction-filtration-comparison test. I first abstract out the various parts of the computer program and then filter out those portions of the program which are not copyrightable. If, arguably, the command codes are considered part of the computer program in the call controller then their sole purpose is to provide access to the functions availa-

decision, but did so for different reasons.[433] In particular, the Tenth Circuit disagreed with the district court and the First Circuit on the use of the Abstraction-Filtration-Comparison analysis in circumstances involving deliberate literal copying stating:

> We conclude that although an element of a work may be characterized as a method of operation, that element may nevertheless contain expression that is eligible for copyright protection. Section 102(b) does not extinguish the protection accorded a particular expression of an idea merely because that expression is embodied in a method of operation at a higher level of abstraction. Rather, sections 102(a) & (b) interact to secure ideas for public domain and to set apart an author's particular expression for further scrutiny to ensure that copyright protection will "promote the. . .useful Arts." Our abstraction-filtration-comparison approach is directed to achieving this balance. Thus, we decline to adopt the *Lotus* court's approach to section 102(b), and continue to adhere to our abstraction-filtration-comparison approach.[434]

Using this analytical framework the Tenth Circuit concluded that the Mitel command codes were largely unoriginal, and to the extent they contained any original expression, that expression was excluded from protection under the scènes à faire doctrine.[435] The Tenth Circuit agreed with the district court that Mitel had used such minimal effort and judgment to select its command codes that they were unoriginal under §102(a), and furthermore the "random and arbitrary use of numbers in the public domain does not evince enough originality to distin-

---

ble in the call controller. Thus, they provide the means to access or operate the program contained in the software (internal citations omitted).

433. *Mitel*, 124 F.3d at 1372.

434. *Mitel*, 124 F.3d at 1372 (10th Cir. 1997) (internal citations omitted). It is interesting to note that the First Circuit in *Lotus V* had criticized the approach taken by the Tenth Circuit in Autoskill, Inc. v. Nat'l Educ. Support Sys., Inc., 994 F.2d 1476 (10th Cir. 1993):

> Our holding that methods of operation are not limited to abstractions goes against *Autoskill*, 994 F.2d at 1495 n. 23, in which the Tenth Circuit rejected the defendant's argument that the keying procedure used in a computer program was an uncopyrightable "procedure" or "method of operation" under § 102(b). The program at issue, which was designed to test and train students with reading deficiencies, *id.* at 1481, required students to select responses to the program's queries "by pressing the 1, 2, or 3 keys." *Id.* at 1495 n.23. The Tenth Circuit held that, "for purposes of the preliminary injunction, . . . the record showed that [this] keying procedure reflected at least a minimal degree of creativity," as required by *Feist* for copyright protection. *Id. As an initial matter, we question whether a programmer's decision to have users select a response by pressing the 1, 2, or 3 keys is original. More importantly, however, we fail to see how "a student select[ing] a response by pressing the 1, 2 or 3 keys," id., can be anything but an unprotectable method of operation.* Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*), 49 F.3d 807, 818-19 (1st Cir. 1995) (footnote omitted) (emphasis added).

435. Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1373-76.

guish authorship."[436] The court found that Mitel's parameter ranges (values in the parlance of the case) contained enough intellectual production, thought and conception to meet the minimal degree of creativity necessary to qualify as an original work of authorship.[437] However, the Tenth Circuit went on to find that this non-arbitrary original expression was unprotectable as scènes à faire because the Mitel values were dictated by external functionality and compatibility requirements of the computer and telecommunications industries.[438]

Thus, the Tenth Circuit found that the technical interface in the *Mitel* case was unprotectable.[439] However, despite the fact that the same result was reached as in *Lotus*, the reasoning in *Mitel* is significantly different. Indeed, the approaches are so different it seems virtually certain that there will be many interfaces for which the two approaches will produce different results. The First Circuit decided not to use the Abstraction-Filtration-Comparison test and instead used §102(b) directly to deny copyright protection for the method of operation at issue. However, the Tenth Circuit chose to apply the Abstraction-Filtration-Comparison test and stated that §102(b) does not "extinguish the protection accorded a particular expression of an idea merely because that expression is embodied in a method of operation at a higher level of abstraction."[440] These two approaches are starkly different. *Lotus* provides a reasonable, bright-line test pursuant to which copyright does not protect the specific words and other subject matters needed to operate a device or program.[441] *Mitel* mandates a very fact dependent approach that may extend copyright protection to essential parts of a method of operation.[442] One of the potential consequences of the *Mitel* approach is that protection may be extended to a portion of a technical interface/method of operation thereby effectively providing *de facto* protection for the entire interface/method of operation.[443] The First Circuit observed

436. *Id.* Mitel's own engineers testified that the command codes contained components that were arbitrary and "real close to random," and that there was no evidence that anyone was trying to "put their mark" on the codes. *Id.*

437. *Id.* at 1375.

438. *Id.* at 1376.

439. *Id.* at 1376.

440. *Id.* at 1372.

441. Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*), 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam).

442. Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366 (10th Cir. 1997).

443. The reason for this is that a commercially viable level of interoperability between two computer programs is generally going to require support for most, if not all, of a particular technical interface. Some standards have mandatory and optional portions; therefore, full support of an interface is not always required. However, a test based on protectable expression is unlikely to correspond to the division between mandatory and optional subject matter. The only situation in which a technical interface would potentially be usable would be the one where those sections that were protected by the owner's copyright were in

that using the Abstraction-Filtration-Comparison test in cases of deliberate literal copying may be misleading because "in instructing courts to abstract the various levels, it seems to encourage them to find a base level that includes copyrightable subject matter that, if literally copied, would make the copier liable for copyright infringement."[444] While this concern did not manifest itself when the Abstraction-Filtration-Comparison test was used in *Mitel*, it seems likely that the application of the *Mitel* test to the *Lotus* facts would have yielded a different result.

The Mitel interface was very rudimentary, yet the Tenth Circuit was still able to find subject matter that met the copyright originality standard.[445] The court's conclusion about originality was probably correct because of the extremely low threshold as enunciated in *Feist*. The only factor that prevented the Mitel interface from having at least some protectable elements was that many of the features of the interface were dictated by external considerations such as standard programming conventions, hardware limitations and various network capabilities.[446] Given the foregoing, it seems unlikely the Tenth Circuit would have concluded that the Lotus 1-2-3 interface did not contain any protectable subject matter. The only remaining question would have been whether that subject matter would have been filtered for reasons such as scènes à faire or other limiting doctrines. With respect to scènes à faire, it is reasonable to assume there would have been few if any external factors that dictated Lotus' choices. Accordingly, under a *Mitel*-like approach there probably would have been sufficient protectable subject matter to prevent direct copying of enough of the Lotus 1-2-3 menus and menu command hierarchy to create a commercially viable compatibility capability.

Since the proper use of a technical interface requires the use of exact words, parameters and data structures, the application of the *Mitel* standard appears likely to prevent the use of many technical interfaces because under *Mitel* the use of certain words, parameters and data structures can infringe copyright. This result seems counterintuitive given the language of 17 U.S.C. §102(b), which provides that "[i]n no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, prin-

---

the optional portion of the interface. A much more likely scenario is that the copyrightable portions of a technical interface will be spread across both the option and mandatory sections of the interface, meaning that anyone who does not have permission from the copyright owner will be, in all likelihood, unable to make any commercial use of that technical interface.

444. *Lotus V*, 49 F.3d at 815.

445. *Mitel*, 124 F.3d at 1374 (finding that the parameter ranges, called "values" in the decision, reveal the existence of intellectual production and conception that reflects at least the minimal degree of creativity required to qualify as an original work of authorship).

446. *Id*. at 1375.

ciple, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."[447] The plain language of §102(b) differs from the meaning ascribed to that section by the Tenth Circuit when it said that "Section 102(b) does not extinguish the protection accorded to a particular expression of an idea merely because that expression is embodied in a method of operation at a higher level of abstraction."[448] Hence, the Tenth Circuit's formulation of Section 102(b) appears to be that in no case, *except if an idea, procedure, process, system, method of operation, concept, principle, or discovery is expressed at a higher level of abstraction,* does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery. This difference may not be significant for some subject matter contemplated by §102(b), such as ideas, concepts, principles and discoveries, where the exact wording of those ideas, concepts, principles and discoveries will not prevent others from using them; however, for processes, systems, and methods of operation that depend on the use of exact wording, the difference is very significant. The First Circuit made this point in *Lotus*, when the court observed that specific words are essential for a computer interface (whether that be a computer-human interface or a technical interface). If some of those specific words cannot be used by potential competitors because of copyright protection, then the use of the entire interface will essentially be denied.[449] Not only does the approach taken by the Tenth Circuit appear to conflict with the plain words of §102(b), but it also appears to conflict with any narrower interpretation of that section that may be suggested by *Baker v. Selden*.[450] Under *Baker v. Selden,* protection is still denied to subject matter that is *necessary* for the use of a method of operation.[451] There is no reason to believe that expression at a lower level of abstraction will not include material necessary for the use of a method of operation at a higher level of abstraction. The key point is that specific words, parameters and data structures are generally required to use any method of operation. If the use of any necessary words, parameters or data structures is prohibited because of copyright protection, then the corresponding method of operation will not be available as a practical matter. Accordingly, use of the Tenth Circuit's *Mitel* test can help support a *de facto* monopoly for sufficiently popular interfaces.

In *Mitel*, the Tenth Circuit adopted an extremely low level of ab-

447. 17 U.S.C. § 102(b) (2008) (emphasis added).

448. *Mitel*, 124 F.3d at 1372.

449. In most cases, partial interoperability between computer and software components will not be commercially viable or will give the original vendor an enormous advantage.

450. Baker v. Selden, 101 U.S. 99 (1879).

451. *Id.* (emphasis added).

straction.[452] Notwithstanding this very low level of abstraction, if the court had not denied copyright protection for other reasons there would have been copyrightable subject matter that could have been used to deny potential competitors practical access to the entire Mitel method of operation since those competitors would not have been able to properly interpret and process many Mitel commands. As a commercial matter, in most cases an inability to fully support a technical interface is going to be tantamount to not being able to support the interface at all. The Tenth Circuit's interpretation of §102(b) is likely to have the effect of nullifying the exclusions contemplated by that section for a very significant number of computer interfaces.[453] It seems unlikely that this was Congress' intent when it drafted §102(b). For computer interfaces that require the precise use of specific words, formats and data structures, the First Circuit's approach in *Lotus* seems to be more in accord with principles first enunciated in *Baker v. Selden* and later codified in §102(b).[454]

iv. Mitek

At about the same time as the Tenth Circuit considered *Mitel*, and only a few months after it had considered similar issues in *Bateman*, the Eleventh Circuit had another chance to examine the scope of copyright protection for technical interfaces. *Mitek Holdings, Inc. v. Arce Engineering Co.*, like *Lotus*, involved a consideration of the scope of copyright protection for menu commands and menu command hierarchies.[455]

Mitek dealt with a wood truss program called ACES, which the plaintiff had developed, and a competing program called TrussPro, which the defendant Arce had developed.[456] These programs were intended to allow truss fabricators to design their own wood trusses without needing an engineer for the design work, thereby allowing fabricators to reduce expenses.[457] The allegedly infringed product originally had been developed by a company called Advanced Computer Engineering Specialties,

452. *Mitel*, 124 F.3d at 1373 (Mitel does not claim copyright in the names of the functions that are accessed by its command codes or in the idea of using four-digit numeric codes to manipulate the functions of a call controller. Rather, Mitel contends that copyright protection extends only to its selection of particular 'values' assigned to the "description" digit of Mitel's codes).

453. While the interface in question was not protected under the Tenth Circuit's test, the arbitrary selection of command names and the very constrained environment in which this interface operated seem to make this result the exception rather than the rule.

454. *See* Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*), 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam).

455. Mitek Holdings, Inc. v. Arce Eng'g Co., Inc., 89 F.3d 1548 (11th Cir. 1996).

456. *Id.* at 1550.

457. *Id.* at 1551.

Inc. ("Advanced").[458] Some years later, Mitek purchased Advanced and as a result gained ownership of ACES.[459] The author of ACES was an Advanced employee named Sotolongo.[460] After Mitek acquired Advanced, Sotolongo left to join Arce.[461] At Arce, Sotolongo was asked to develop "from scratch" another wood truss layout program.[462] That effort resulted in the creation of the TrussPro program. Shortly after the commercial release of Trusspro, Mitek filed an action alleging copyright infringement.[463]

The district court in this case held that there was no copyright infringement. One of the primary reasons for this finding was the district court's ruling that the menu and submenu command tree structure in the ACES program was an uncopyrightable process because it mimicked the way a draftsman would design a roof truss plan by hand.[464] The Eleventh Circuit agreed with this finding. In commenting on the district court ruling, the Eleventh Circuit said:

> Mitek seems to misapprehend the fundamental principle of copyright law that copyright does not protect an idea, but only the expression of the idea. The idea-expression dichotomy is clearly set forth in 17 U.S.C. §102(b), which by its express terms prohibits copyright protection for "any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work." 17 U.S.C. 102(b). Were we to grant copyright protection to Mitek's user interface, which is nothing more than a process, we would be affording copyright protection to a process that is the province of patent law. As the Federal Circuit stated, "Patent and copyright laws protect distinct aspects of a computer program." *Atari Games Corp. v. Nintendo of America, Inc*., 975 F.2d 832, 839 (Fed.Cir.1992). Patent law "provides protection for the process or method performed by a computer in accordance with a program," whereas copyright protects only "the expression of that process or method." *Id*. If, however, the patentable process and its expression are indistinguishable or inextricably intertwined, then "the process merges with the expression and precludes copyright protection." *Id*. at 839-40. Such is the case with the menu and the submenu command tree structure of the ACES program.[465]

The Eleventh Circuit went on to comment on *Lotus* and indicated that, unlike the First Circuit, "we need not decide today whether a main

458. *Id.* at 1552.
459. *Id.*
460. *Id.* at 1551.
461. *Mitek Holdings,* 89 F.3d at 1552.
462. *Id*.
463. *Id.* at 1152-53.
464. *Id.* at 1556.
465. *Id.* at 1556-57 n.19.

menu and submenu command tree structure is uncopyrightable *as a matter of law*."[466] This observation is somewhat different than the statement made by the same court in *Bateman*, when it said "[i]t is an incorrect statement of the law that interface specifications are not copyrightable as a matter of law."[467] In *Mitek*, the Eleventh Circuit's potential disagreement with the First Circuit is more narrowly focused since the subject matter in *Mitek* is virtually identical to that in *Lotus*. Hence the differences between the First Circuit and the Eleventh Circuit appear to be substantive and not a function of the subject matter in the respective cases that those courts considered.

The outcome in *Mitek* is somewhat similar to the outcome in *Mitel*. Ultimately, the plaintiff in each case was unable to succeed because its interface did not have enough protectable expression to support a finding of substantial similarity.[468] However, the particular doctrines used in each case to eliminate potentially protectable subject matter differed. The decision to classify the menus and menu command hierarchies in *Mitek* as a process rather than a method of operation was also different. Nothing in the *Mitek* decision suggests that the ACES menu command hierarchy could not have been classified as a method of operation, as was done for the menu commands and menu command hierarchy in *Lotus* or the command interface in *Mitel*. The decision to classify the menus and menu hierarchy as a process is probably reasonably unusual and due to the fact that the ACES menu commands so closely tracked the actions an actual draftsman would take in creating a roof truss plan. However, since copyright protection for both processes and methods of operation is excluded under §102(b), the classification of the menu command hierarchy as a process rather than a method of operation should not affect the application of that section.

---

466. *Id.* at 1557. The Eleventh Circuit went on to observe:

> Even were we to conclude that section 102(b) does not prohibit the ACES main menu and submenu command tree structure from being entitled to copyright protection, MiTek would not prevail on this issue. This feature of the ACES programs is unoriginal and not entitled to copyright protection. The look of the ACES program is basically industry standard for computer aided-design ("CAD") programs, with the menu bars running across the top and the right, and the large work area occupying most of the screen. In addition, based on the district court's conclusion that the ACES programs "mimic the steps a draftsman would follow in designing a roof truss plan by hand," a conclusion with which we find no fault, the structure of the menu and submenu command tree of the ACES programs tracking that approach is unoriginal and uncopyrightable. The logical design sequence is akin to a mathematical formula that may be expressed in only a limited number of ways: to grant copyright protection to the first person to devise the formula effectively would remove that mathematical fact from the public domain. The merger doctrine prohibits such an appropriation. *Id.* at 1557 n.20.

467. Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1547 (11th Cir. 1996).

468. *Mitek Holdings,* 89 F.3d 1548; *See also* Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366 (10th Cir. 1997).

As in *Mitel*, the court found that there was no substantial similarity largely because of the Filtration phase of the Abstraction-Filtration-Comparison test. In *Mitel*, the Filtration phase relied on scènes à faire and lack of originality.[469] In *Mitek*, the Filtration phase relied on merger and lack of originality.[470] With respect to originality, the court observed that the look of the ACES program was basically industry standard for computer aided-design programs.[471] With respect to merger, the court said the logical design sequence implemented by ACES was akin to a mathematical formula that may be expressed in a limited number of ways.[472] Consequently, the court was concerned that a grant of copyright protection in such circumstances would effectively remove the logical design sequence from the public domain.[473]

In general, *Mitek* is similar to *Mitel* in that it uses the Abstraction-Filtration-Comparison test and accordingly appears open to the possibility that certain expression that may be necessary for the use of a process or method of operation may be protected. However, while *Mitek* is not supportive of the *Lotus* approach, the court did not explicitly reject that approach. *Mitek* seems destined to be less relevant than cases like *Mitel* and *Lotus*. First, *Mitek* did not take a definitive position on the protection to be accorded as a matter of law to potentially copyrightable subject matter in menus and menu command hierarchies. Second, any subsequent courts, if they want, should be able to limit the application of this case to circumstances in which an interface closely tracks a process – a situation that is probably much less likely to occur than cases in which an interface is a method of operation.

Since the menus and menu command hierarchy in this case were not found to have any copyrightable subject matter, this case does not provide any insight into how the Eleventh Circuit would rule in a case dealing with non-filterable expression. While *Mitek* is like *Mitel* in that it suggests that essential parts of a process, and by extension a method of operation, can contain copyrightable expression, the court did not take a definitive stand, thereby giving other courts further opportunity to distinguish this case if they so choose. In this regard, *Mitel* is more significant because the court in that case indicated how it intends to deal with interfaces that contain copyrightable expression. Since the interface in *Mitel* did not contain protectable subject matter, the Tenth Circuit did not need to address the issue of whether copyright protection should be denied to methods of operation as a matter of law, but instead the Tenth Circuit chose to state its position on this broader issue. Ultimately,

469. Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1373-76 (10th Cir. 1997).
470. *Mitek Holdings,* 89 F.3d at 1558.
471. *Id.* at 1557.
472. *Id.*
473. *Id.*

Mitek does not provide significant guidance in respect to copyright protection for technical interfaces since on its narrow facts it deals with an uncommon circumstance; and, in respect to the broader issues, the Eleventh Circuit chose to remain silent.

v. Baystate

At about the same time as the *Lotus*, *Mitel, Bateman,* and *Mitek* cases were being decided by the First, Tenth, and Eleventh Circuits, a district court in the First Circuit decided a case that has been touted as having significance for the scope of copyright protection for technical interfaces.[474] In *Baystate Technologies, Inc. v. Bentley Systems, Inc.*, the district court for Massachusetts decided that certain data structures in a computer-aided design program, including their names and organization, were not protectable under copyright.[475] It has been suggested that these program elements are indistinguishable from a larger class comprised of names, parameters, formats, and structures that must be employed to invoke the functionality of various operating systems, computer languages and application programs and that *Baystate* is supportive of the proposition that using or copying these labels and structures for compatible works is permitted under copyright law.[476] While this proposition may be true, it seems unlikely that *Baystate* will be helpful in supporting it. Upon examination, the reasoning in *Baystate* is at some points mystifying and at other points simply wrong.

The factual setting in *Baystate* is reasonably complex because of the use of a common third-party outsourced developer, and also because Baystate was not the original developer of the allegedly infringed program. However, these complexities do not ultimately change the copyright analysis, and, accordingly, the case can be distilled to the following facts. The plaintiff, Baystate Technologies, was the owner of a computer-aided design product called CADKEY.[477] The defendant, Bentley Systems, was the owner of a software product called the Microstation Translator.[478] The Microstation Translator was an add-on to the defendant's Microstation CAD product.[479] The function of the Microstation Translator was to convert files in CADKEY format into a format understood by the Microstation product.[480] To accomplish this task the Microstation Translator needed to read files in the CADKEY format and properly

474. Zimmerman, *supra* note 306.
475. Baystate Technologies, Inc. v. Bentley Sys., Inc., 946 F. Supp. 1079 (D. Mass 1996).
476. Zimmerman, *supra* note 306, at 9.
477. *Baystate Technologies*, 946 F. Supp. at 1082.
478. *Id.*
479. *Id.*
480. *Id.*

parse them into their various components.[481] This was done by translating data structure definitions from the CADKEY product into the corresponding native data structures used by the Microstation CAD product.[482] Thus, the Microstation Translator used some CADKEY data structures and, in particular, used the various names of those data structures and their sub-components.[483]

The first of a number of curious statements made by the *Baystate* court is that "because the data structures at issue in this case do not bring about any result on their own, they are copyright protected, if at all, only as a part of the whole computer program."[484] According to 17 U.S.C. §101 a computer program is "a set of statements or instructions to be used *directly or indirectly* in a computer in order to bring about a certain result."[485] Data structure definitions are directives to a compiler and ultimately to a computer and operating system about memory organization in support of operations to be performed by a particular computer program.[486] At the very least it would seem that data structures definitions are used indirectly to bring about a certain result and should be accorded copyright protection in the same manner as any other statement or instruction in a computer program. Later, the court states that, "data structures are not, by themselves, executable, i.e. a computer cannot read data structures and perform any function."[487] From a computer science and copyright perspective this statement makes little sense.[488] Computers cannot read anything other than zeros and ones, but that does not mean only machine code is protected under copyright law. As a practical matter, any computer program of any degree of sophistication will consist of executable instructions and other non-executable components that are nonetheless required for the operation of the program. At least one other district court has held that data structures are included within the definition of a "computer program."[489]

---

481. *Id.*

482. *Id.*

483. *Baystate Technologies*, 946 F. Supp. at 1085.

484. *Id.* at 1086.

485. 17 U.S.C. § 101 (emphasis added).

486. The first step in designing a computer program is often a determination of the data structures required to best achieve the desired result. Once this determination has been made, the rest of the program is designed and developed accordingly.

487. *Baystate Technologies*, 946 F. Supp. at 1085.

488. *See*, Marci Hamilton & Ted Sabety, *Computer Science Concepts in Copyright Cases: The Path to a Coherent Law*, 10 HARV. J.L. & TECH. 239, 247-49 (1997) (stating that computer programs can be divided into three basic parts, one of which is the "data processing section - the heart of any program"; and, "[i]nside the data processing section lies the collection of algorithms and data structures that actually perform the computations that users demand").

489. Positive Software Solutions, Inc. v. New Century Mortgage Corp., 259 F. Supp. 2d 531, 535 (N.D. Tex. 2003).

In its analysis, the district court noted that the First Circuit held that although the authors of Lotus 1-2-3 "made some expressive choices in choosing and arranging the Lotus command terms, we nonetheless hold that that expression is not copyrightable . . . The 'expressive' choices of what to name the command terms and how to arrange them do not magically change the uncopyrightable menu command hierarchy into copyrightable subject matter."[490] The district court observed that there was evidence to suggest that the CADKEY data structures were independently created and were original expression. However, in reliance on *Lotus* and applying merger and scènes à faire, the court concluded that the choices made in selecting the CADKEY data structure names could not make the uncopyrightable data structures copyrightable.[491]

The district court's merger analysis is abbreviated and it is unclear to what extent the court relied on merger in deciding the case. After its merger analysis, the court then considered the applicability of scènes à faire. As discussed in *Positive Software Solutions, Inc. v. New Century Mortgage Corp.* and *Zimmerman*,[492] the district court reversed the externalities consideration in the scènes à faire analysis. In particular, the district court made the following observation:

> In this case, the court concludes that the selection and organization of the elements in the data files is dictated mainly by external factors. *The product being developed* is a data translator that is designed to "read" the data files of CADKEY. The process of "reading" the CADKEY data files requires that the elements contained within the data structures of the Translator be organized in the same manner as the elements in the data structures of CADKEY. Without such compatibility, the Translator would not function because it would "misread" the CADKEY data files.[493]

It is apparent from the subsequent discussion that the phrase "product being developed" can only refer to the allegedly infringing program and not the allegedly infringed program.[494] This type of scènes à faire

---

490. Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807, 816 (1st Cir. 1995).

491. *Baystate Technologies*, 946 F. Supp. at 1088-89.

492. *Positive Software Solutions*, 259 F.Supp. 2d at 535 n.9; *See* Zimmerman, *supra* note 306, at 16-17.

493. *Baystate Technologies*, 946 F. Supp. at 1088 (emphasis added).

494. The court observes:

> Significant differences between the names and organization of the names used in the "target product" and the translator would be inefficient for the programmer. For that reason, the data structure names in Infotech's MODES product had to be similar to the data structures in the Part File Toolkit documentation because the computer programmer needed to refer to the documentation in the process of creating and manipulating the CADKEY "read" capability of MODES. *Id.* at 1088-89.

The court further observes:

> With regard to industry-wide standards, Scott Taylor, an apparently objective and neutral witness, shed at least some light. Mr. Taylor developed many translators,

analysis is obviously incorrect. The proper approach is to consider only the allegedly infringed program and determine whether the relevant portions of that program were dictated by external factors. In the context of *Baystate*, this would have meant examining the CADKEY data structures and determining whether Baystate's choices in developing those data structures were dictated by any external factors. Instead, the district court examined the allegedly infringing program and determined that its use of the CADKEY data structures was an externality that was dictated by a requirement to interoperate with CADKEY. This type of analysis may have had some value in a fair use analysis, but is entirely incorrect in a scènes à faire analysis. The fact that the CADKEY data structures subsequently became an externality for another program cannot retroactively make those data structures uncopyrightable. In its scènes à faire analysis the district court also seems to have been influenced by "industry standards." Specifically, the court observed that an apparently neutral witness testified that he had developed many translators, including one that could read CADKEY-formatted files, and it was his practice to use, at least to some extent, the target product's data structure names and organization.[495] In relying on this evidence, the court seems to be using other instances of potential copyright infringement to support a finding that the data structures at issue are not copyrightable. It is unclear how the copying of different data structures or other infringements of the data structures at issue can affect their copyright status. For these reasons, *Baystate* seems to provide little useful analysis and is of virtually no value as precedent when determining whether data structures and by extension technical interface structures and names are protected under copyright.

vi. Positive Software

A number of years after *Baystate*, another district court in the Fifth Circuit considered similar questions regarding the scope of copyright protection for data structures. This district court, having the benefit of the *Baystate* reasoning, reached a different conclusion about the copyrightability of data structures. In *Positive Software Solutions, Inc.*

including one which could read CADKEY. It was his practice to use, at least to some extent, both the target product's data structure names and its organization of the data structures. Hence, when he created a data translator which could "read" CADKEY files, he used the CADKEY file names and the organization of data structures as they were described in the Part File Toolkit documentation. Walter Anderson testified that he used the same method in creating a translator to read AutoCAD files. On the basis of relatively limited evidence of the CAD industry standards, and after applying all of the relevant copyright principles, this Court concludes, therefore, that the data structure names and the organization of those names are not protected expression under the copyright laws. *Id.* at 1089.

495. *Id.* at 1089.

v. New Century Mortgage Corp., the District Court for the Northern District of Texas considered whether data structures originally developed for one program can be used by another program seeking to either operate with the first program or utilize data formatted for the first program.[496] The first program in this case was a product developed by Positive Software called LoanForce.[497] LoanForce was designed to provide automated support for the mortgage loan business, and, in particular, was designed to interact with a database to store and retrieve information relevant to potential borrowers.[498] New Century was in the mortgage business and had licensed LoanForce from Positive Software for use in New Century's business.[499] The license was on a subscription basis, meaning New Century paid an annual license fee that allowed it to use the LoanForce product for that particular year.[500] Eventually, New Century decided it wanted to save the money it was spending on LoanForce and New Century had its own similar product developed.[501] This development occurred while New Century was still using LoanForce.[502] The New Century replacement product was developed in phases and an interim version of the replacement product, known as LoanTrack-1, was developed first.[503] This interim version did not provide the full functionality of LoanForce, and it needed to work in conjunction with LoanForce to provide all of the capabilities of LoanForce.[504] At the same time, New Century was working on a final version of its product, called LoanTrack-2, which would provide the full functionality of LoanForce without having to rely on it in any way.[505] Positive Software became aware of LoanTrack-1 and LoanTrack-2 and concluded that New Century was making improper use of its intellectual property rights.[506] As a result, Positive Software sought a preliminary injunction to prohibit New Century from continuing to infringe its intellectual property rights in LoanForce.[507]

In deciding an application for a preliminary injunction, the district court considered whether certain LoanForce data structures were copy-

496. Positive Software Solutions, Inc. v. New Century Mortgage Corp., 259 F.Supp. 2d 531 (N.D. Texas 2003).

497. *Id.* at 533

498. *Id.*

499. *Id.*

500. *Id.*

501. *Id.*

502. *Positive Software Solutions*, 259 F.Supp. 2d at 533.

503. *Id.*

504. *Id.*

505. *Id.* at 534.

506. *Id.*

507. *Id.*

rightable.[508] LoanForce had been designed to interact with a database to store and retrieve borrower information.[509] LoanForce performed its data transfer operations using statements written in Structured Query Language or SQL.[510] The first issue for the court was to determine whether the LoanForce SQL data structures were copyrightable subject matter, and the court had little trouble concluding they were.[511] As in *Baystate*, the court started with the statutory definition of "computer program," however, this court concluded that "the SQL Data Structures here are a set of statements to be used indirectly in a computer in order to bring about a certain result. Accordingly, the SQL Data Structures are proper subject matter for copyright protection."[512] In this regard, the court correctly observed that the definition of "computer program" includes both direct and indirect steps used to bring about a certain result.[513] Further, the court did not limit the definition of "computer program" to only those steps that are actually executed or bring about a direct result on their own.[514] The court reached this conclusion notwithstanding its consideration of *Baystate*. In discussing *Baystate*, the court observed that the expression in this case went well beyond the "minimal degree of creativity" or "minimal creative spark" required by *Feist* and that there was no indication that the granting of copyright protection to the data structures at issue would grant a monopoly over any algorithm.[515] The district court also indicated that it did not subscribe to the view taken by the court in *Baystate* that the role of creative expres-

508. *Positive Software Solutions*, 259 F.Supp. 2d at 535.

509. *Id*.

510. *Id*.

511. *Id*.

512. *Id*.

513. It should be noted that SQL or Structured Query Language was used to define the data structures in Positive Software. SQL is a computer language that is designed to store, manipulate, and retrieve data stored in relational databases. SQL is a declarative programming language. A declarative programming language is a high-level language that describes a problem rather than describing a solution to a problem. By contrast, an imperative programming language describes how to obtain a solution. SQL does not describe how to find data in a database, but instead describes criteria for finding that data. The data structures in a more traditional language, such as C or Java, are also declarative. The purpose of the data structures in C or Java is to declare or describe the organization of data that will be used by the imperative portions of a program. These data structures in and of themselves do not describe how to solve a problem but are instead used in helping to solve that problem. Because SQL was used, the data structures in *Positive Software* probably looked more like the imperative computer program instructions typically seen in C or Java. However, based on the reasoning in the judgment, it does not appear that the fact that the data structures in Positive Software were defined using SQL rather than data structure syntax such as that found in C or Java affected the outcome.

514. *Positive Software Solutions*, 259 F.Supp. 2d at 535.

515. *Id.* at 535 n.6.

sion in developing data structures was of little importance.[516]

Having concluded that the SQL data structures were copyrightable, the court needed to determine whether there had been actionable copying. The court found that New Century's replacement product, LoanTrack-1, contained substantial verbatim or near verbatim copying of the SQL statements from LoanForce.[517] Therefore the question of factual copying was easily resolved. The remaining question was whether the copied material supported a finding of substantial similarity. For the purposes of this analysis, the district court used a modified Abstraction-Filtration-Comparison method.[518] Because of the verbatim copying, the court did not undertake an Abstraction phase.[519] Instead, the court proceeded directly to the Filtration phase to determine whether any parts of the data structures should be removed prior to the Comparison phase.[520] The district court identified scènes à faire as the most relevant limiting doctrine, but then dismissed the applicability of that doctrine.[521] In doing so, the district court again referred to *Baystate* and rejected the approach taken there noting that "[the] commercial compatibility argument is more in the nature of a fair use argument, rather than an argument that certain aspects of the copyrighted work were dictated by market factors and were thus unprotectable."[522] The court went on to find that there was no evidence to suggest that the design of the SQL data structures, such as the organization of data into tables, the selection of column elements for the tables, the names, data types, or sizes of the column elements, were dictated by external factors.[523] As a result, the court did not filter any of the SQL data structures.[524] Having completed the Filtration phase, the court did a comparison and found that the amount of near verbatim copying of significant portions of the SQL data structures supported a finding of substantial similarity between LoanTrack-1 and LoanForce.[525]

In *Baystate* and *Positive Software*, two district courts reached opposite conclusions about the copyrightability of data structures. In one case, a court expressed the view that the data structures in question were neither a substantial nor significant part of the whole copyrighted

516. *Id*. at 535 n.5.
517. *Id.* at 535-36.
518. *Id*.
519. *Id.* at 536.
520. *Positive Software*, 259 F.Supp. 2d at 536.
521. *Id*.
522. *Id.* at 356 n.9.
523. *Id.* at 356.
524. *Id*.
525. *Id.* at 537.

work.[526] In the other case, a court found that even given a sliding scale with narrower protection for functional works, the significance of the data structures to the program as a whole supported a finding of substantial similarity.[527] While it is not possible, without a detailed examination of the works in question, to ascertain the relative importance of the particular data structures to their respective programs, it is worth noting that the data structures performed roughly the same function in each case. In one instance, the program stored design data in a file for retrieval and processing by a CAD tool; in the other case, the program stored customer/prospect data in a database for retrieval and processing by a customer management tool. Thus both programs stored raw data that was to be retrieved and utilized by the applicable program. When designing a program, software developers tend to view data definition, data processing and data flow identification as significant steps. Hence, data structures are often critical in determining the overall structure and design of a program.[528] It seems reasonable to conclude that a typical software developer will probably ascribe more value to the data structures within a program than did the court in *Baystate*. In support of its conclusions, the court in *Positive Software* noted that the copyright requirement for creativity is quite low and the data structures in question could not be characterized as generic.[529] Given the general importance ascribed to data structures by software developers, it would seem that the court in *Positive Software* reached the correct conclusion about the copyrightability of data structures.

The *Positive Software* court also correctly observed that scènes à faire is applied by considering the allegedly infringed program and not the allegedly infringing program. This difference is crucial because when a program is being designed to be compatible with another program, the program being designed for compatibility will almost always need to use the same data structures, APIs, and protocols as the program with which compatibility is being sought. For a program seeking compatibility, the use of these program elements is always going to be an

526. Baystate Technologies, Inc. v. Bentley Sys., Inc., 946 F.Supp. 1079, 1089 (D. Mass 1996).

527. *Positive Software Solutions*, 259 F.Supp. 2d at 537.

528. *See*, CMAX/Cleveland, Inc. v. UCR, Inc. 804 F.Supp. 337, 344 n.3 (M.D. GA. 1992). The court in this case observed that:

> The determination of how to store data in files is a crucial element in the design process of a computer software system. The layout or blueprint for data storage is the foundation upon which a computer system is built, and is the result of a creative thought process. For example, when creating a file, a system designer must make numerous decisions concerning the material to be included in the file, the order of that material and how that material can be accessed and used by the system.

529. *Positive Software Solutions*, 259 F.Supp. 2d at 535 n. 5.

externality. If scènes à faire was applied in the manner it was in *Baystate*, then the result would be pre-ordained and the elements required for compatibility would always be filtered. However, as was observed in *Positive Software*, while this type of analysis may be relevant for a fair use analysis, it is incorrect when used to determine the copyrightability of a program with which compatibility is being sought. Instead, the pre-existing program must be considered on its own without reference to subsequent programs that seek to interoperate with it. If the data structures in such a pre-existing program are to be filtered because of externalities, only those externalities dictated to the pre-existing program can be used. Such externalities can never arise from a subsequent program seeking compatibility with the pre-existing program.

For these reasons, the analysis in *Positive Software* appears to be much stronger than that in *Baystate*, and it seems likely that subsequent courts considering these matters will be much more likely to adopt the reasoning in *Positive Software* when considering copyright protection for data structures.

vii. Engineering Dynamics

One of the cases that the *Positive Software* court needed to consider was *Engineering Dynamics, Inc. v. Structural Software, Inc*.[530] Since the Fifth Circuit Court decided *Engineering Dynamics*, the court in *Positive Software* needed to ensure that its reasoning was consistent with the Fifth Circuit's decision in that case which held that copyright could apply to certain user interface input/output formats. *Engineering Dynamics* is interesting for two reasons. First, as a circuit court decision, it is a stronger precedent, and second, its factual setting contains elements often found in cases dealing with copyright protection for computer interfaces and other elements often found in cases dealing with copyright for data structures. As such, *Engineering Dynamics* raises interesting questions about how these issues interrelate.

In *Engineering Dynamics*, the plaintiff, Engineering Dynamics, Inc. ("EDI"), brought a claim of copyright infringement against the defendant Structural Software ("Structural") alleging that a Structural product called StruCAD infringed copyright in an EDI product known as SACS IV.[531] The technical interface in this case included data structures that represented 80-column fields, which recorded various data items relevant to the analysis of environmental and other forces on physical structures.[532] These data structures had originally been recorded on punch

530. *See*, *Engineering Dynamics, Inc. v. Structural Software, Inc.*, 26 F.3d 1335 (5th Cir. 1994).

531. *Id.* at 1338.

532. *Id.*

cards, but were subsequently moved to magnetic storage devices.[533] In the parlance of the case, the data structures were referred to as "input formats" or "cards."[534] As described by the court, the input formats were a series of words and a framework of instructions that acted as prompts for the insertion of relevant data.[535]

EDI had an interesting challenge in making its claim because EDI had previously successfully defended a lawsuit filed against it in which it had been alleged that some of the input formats used in an earlier version of the EDI product (SACS II) infringed those of another competitor's product.[536] In that earlier case, the court ruled that the nine input formats at issue, including their structure, sequence and organization, were not copyrightable.[537] Because of this earlier ruling, EDI did not claim copyright protection for any individual input formats, but instead claimed copyright protection based on the structure, sequence and organization of its input formats as a whole.[538]

Accepting this approach, the Fifth Circuit ruled that the input formats were protectable expression.[539] Specifically, the court held that "EDI has proved original expressive content in the selection, sequence and coordination of inputs."[540] The court observed that there were other programs that performed the same types of analyses, but had dissimilar interfaces.[541] The Fifth Circuit also drew support from the decision of the district court in *Lotus*. The Fifth Circuit's finding that the input formats as a whole had sufficient originality to meet the copyright standard is almost certainly correct. However, the First Circuit subsequently overruled the district court decision in *Lotus*.[542] In that case, the First Circuit Court held that the district court had inappropriately limited the Lotus 1-2-3 method of operation to an abstraction.[543] The First Circuit held that methods of operation are not limited to abstractions and if specific words are essential to operating something then those words are

533. *Id.*

534. *Id.*

535. *Id.*

536. Synercom Tech., Inc. v. Engineering Dynamics, Inc., 462 F.Supp. 1003 (N.D. Tex. 1978).

537. *Id*.

538. *Engineering Dynamics*, 26 F.3d at 1342 ("EDI makes a different claim that several dozen input formats taken together form a copyrightable work, because they represent but one of many ways of expressing a mode of computerized structural analysis"). Based on this distinction, the court ruled that this action was distinguishable from the earlier Synercom case.

539. *Id.*

540. *Id.* at 1346.

541. *Id.*

542. Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*), 49 F.3d 807 (1st Cir. 1995).

543. *Id.* at 816.

part of a method of operation and are not protectable.[544] If the Fifth Circuit had used this approach instead of the approach taken by the district court, the results in *Engineering Dynamics* would probably have been quite different.

The Fifth Circuit never considered whether the EDI input formats were a part of a method of operation – at least not in the manner contemplated by the First Circuit.[545] There is, however, enough of a factual description in the decision to reasonably conclude that the input formats were probably a part of a method of operation under the criteria used by the First Circuit in *Lotus*. As was discussed earlier, the First Circuit held that a method of operation is "the means by which a person operates something, whether it be a car, a food processor, or a computer."[546] The EDI input formats or cards were described in the following ways:

> The purpose of the SACS input formats is to mediate between the user and the program, identifying what information is essential and *how it must be ordered to make the program work.*[547]
>
> The input and output formats for SACS IV are quasi-textual; while they guide the user in performing a series of sophisticated structural analyses, they consist of a series of words and a framework of instructions that act as prompts for the insertion of relevant data.[548]
>
> [Structural] asserts that the data formats are merely a template that enables an engineer to use his tool, the computer.[549]

According to the First Circuit, once an interface has been determined to be a method of operation, then the expressive choices made in the creation of the components of that interface that are necessary for the use of the method of operation cannot transform the uncopyrightable method of operation into copyrightable subject matter.[550] As First Circuit further observed, the fact that the interface could have been expressed in a number of different ways is irrelevant.[551] One factual aspect of *Engineering Dynamics* that differs from *Lotus* is that the data

---

544. *Lotus V*. 49 F.3d at 816.

545. *Engineering Dynamics*, 26 F.3d at 1346. The court did consider whether the user interface was a process or method in what the court characterized as a *Baker v. Selden* argument. In concluding that the input formats were not a process or method the court observed that "[t]he question is whether the utilitarian function of the input formats, which ultimately act like switches in the electrical circuits of the program, outweigh their expressive purpose so as to preclude copyright protection." *Id.*

546. *Lotus V,* 49 F.3d at 815.

547. *Engineering Dynamics*, 26 F.3d at 1346 (emphasis added).

548. *Id.* at 1342.

549. *Id.* at 1345. This was a characterization suggested by the defendant. While the suggestion that the input formats are mere templates is probably incorrect, the assertion that the input formats are used to operate the program is consistent with statements made by the court.

550. *Lotus V,* 49 F.3d at 816.

551. *Id.*

structures in question were almost certainly used in other parts of the SACS IV program. However, given that the data structures were such an integral part of the method of operating SACS IV, it seems reasonable to conclude that, under *Lotus*, copyright protection would have been denied at least to the extent the data structures were used within the method of operation, since the method of operation could not have been properly used without them. If such an approach is taken in subsequent cases, it will have significant implications for the copyright analysis of dynamic linking and inter-process communication. Complex data structures are often used in both dynamic linking and inter-process communication to describe the parameters exchanged by linking programs or the messages exchanged by communicating processes.

Having determined that there was sufficient originality to conclude that the input formats contained copyrightable expression, the Fifth Circuit remanded the case back to the district court to perform a Filtration analysis on the input formats.[552] Therefore, this case did not definitively determine whether the input formats contained protectable subject matter, however, the final result of the Filtration analysis is not important. The important finding was that the SACS IV input formats could have been protected under copyright. Accordingly, *Engineering Dynamics* is similar to *Mitel* and differs from *Lotus*, which probably would have denied copyright protection.

viii. CMAX/Cleveland

CMAX/Cleveland, Inc. v. UCR, Inc. is another case with similar facts to *Engineering Dynamics* and *Positive Software* and the district court in *CMAX/Cleveland* reached largely the same conclusions as the courts in *Engineering Dynamics* and *Positive Software*.[553] In *CMAX/Cleveland* the defendant copied various screen displays, report formats, file layouts, file names and transaction codes used in the plaintiff's program.[554] The defendant had been a licensee of the plaintiff and had decided it wanted to develop a program of its own to save license fees.[555] The evidence suggested that the defendant developed its program by copying the design of the plaintiff's program, including side-by-side comparisons of the respective programs and detailed reviews of the plaintiff's source code.[556] There was also evidence that the defendant was in breach of its license agreement and had negotiated in bad faith in con-

552. *Engineering Dynamics*, 26 F.3d at 1347.
553. *See* CMAX/Cleveland, Inc. v. UCR, Inc., 804 F. Supp. 337 (M.D. Ga. 1992).
554. *Id*.
555. *Id*.
556. *Id*.

nection with a possible site license for the plaintiff's program.[557]

As in *Positive Software* and *Engineering Dynamics*, the Abstraction-Filtration-Comparison test was used as the analytic framework, although, as in *Positive Software*, there was virtually no Abstraction analysis. Like the earlier cases, the court concluded that the file formats were expressive because "Computermax designed its file structures from a myriad of possible options and alternatives."[558] The file formats were not filtered because they were not dictated by externalities. The court reached similar conclusions for the copied screen displays, report layouts, and transaction codes.[559]

Thus *CMAX/Cleveland* does not provide any new insight into the approaches taken by courts in these types of scenarios. It is, however, interesting to speculate about the outcome under a *Lotus*-like analysis. Unlike *Engineering Dynamics*, where a *Lotus*-like approach probably would have led to a different result, it is not clear that a *Lotus*-based analysis would have changed the outcome in this case. The reason for this is that it is not clear that the file structures in *CMAX/Cleveland* are part of a method of operation. In this case, the file formats were essentially a description of data stored in a repository or a database that could be retrieved, analyzed and processed by a user of the program. However, it is not clear that these file formats were part of the means by which the program was operated. The file formats/data structures seem to have been used in a similar manner to those in *Positive Software* and differently from the way the file formats/data structures were used in *Baystate* and *Engineering Dynamics*. In those cases, the files and file formats were used to input the data to be processed. The provision of data to those programs was a key step in their operation. In contrast, in *CMAX/Cleveland* and *Positive Software* the file formats appear to describe data stored in databases used by the respective programs. This data appears to have been originally supplied to the programs by some other mechanism that did not use the data formats at issue.[560] While it is still possible that a court might find that such database or repository formats are

557. *Id.* at 343-44. The evidence suggests that the defendant negotiated a price for a site license that it never intended to purchase and that the negotiations were done to gain time to continue to study and copy the plaintiff's program.

558. *Id.* at 355.

559. *CMAX/Cleveland*, 804 F. Supp. at 355.

560. *Id.* Based on the limited description in the case, it seems possible that some of the material at issue may have been part of a method of operation. For example, the court observed that the defendant copied the transaction codes so employees who were familiar with the plaintiff's system would not have to relearn different codes to use the defendant's system. Based on this observation, it appears that the transaction codes may have been part of a menu command system. If the transaction codes were part of a menu command system, then under a Lotus-based analysis, the transaction codes may not have been protected under copyright law.

part of a program's method of operation, the probability of such a finding seems less likely than that for input formats such as those in *Baystate* and *Engineering Dynamics*.[561]

Thus, under a *Lotus*-based analysis, it seems likely that the results in *CMAX/Cleveland* would have been largely the same. Assuming the data formats in question were not part of a method of operation, they would have qualified as protected expression and would have supported a finding of substantial similarity and copyright infringement (as was ultimately determined by the *CMAX/Cleveland* court). It also seems likely that a similar result would have been reached for the various screens and report formats. The one area where the results may have differed from the actual results was in the transaction codes. Based on the limited description in the decision, it is not possible to make a definitive determination. However, if the transaction codes were used in the operational interface of the program then under *Lotus* there would have been a greater chance they would not have been protected because they may have been a necessary part of the method of operation for the program. If the transaction codes were not used in an operational interface then the outcome would probably be the same as in the actual case.

ix. Lexmark v. Static Control

The Sixth Circuit Court of Appeals has also considered the scope of copyright protection for methods of operation. In *Lexmark International,*

561. It is also interesting to consider whether output formats such as those in *Baystate* and *Engineering Dynamics* are part of a method of operation under a *Lotus*-based analysis. Under *Lotus* it appears that subject matter will not be protected if it is part of the means by which a program is operated. While it is reasonably easy to conclude that the manner in which information is provided to a program is part of the means of operating that program, it is not as easy to conclude that the manner in which information is output by a program is part of the means by which that program is operated. It can certainly be argued that getting results from a program is critical to operating that program. Applying that rationale, the output of results could qualify as part of the method of operation for a program. However, as a practical matter, the ability to read input formats seems to be much more important from a commercial perspective. As illustrated in *Baystate* and *Lotus*, the ability to read input formats is very important for new entrants in a market to enable them to offer commercially viable alternatives to an incumbent product. In both *Baystate* and *Lotus*, each of the defendants had their own file formats and menu hierarchies and still felt there was a commercial requirement to read existing files or macros in a competitor's format. In these cases, the main issue was not the need to generate output in a particular existing format, but instead the need to read input in a particular existing format. Once that existing input data has been read and processed, it can be output in the new entrant's native format. Thus, while output formats may be determined to be part of a method of operation, output formats do not seem to be as commercially significant as input formats. Further, if output formats do not qualify as part of a method of operation, it seems likely that commercial entities will still be able to create competitive products without having to rely on competitors' output formats.

Inc. v. Static Control Components, Inc., the Sixth Circuit Court of Appeals considered methods of operation in the context of lock-out codes used to enable or prevent operation of a computer printer with particular toner cartridges.[562]

This case was initiated by the well-known printer manufacturer Lexmark against a company supplying microchips for use in Lexmark-compatible toner cartridges.[563] Lexmark marketed two types of toner cartridges for its printers.[564] "Prebate" cartridges were sold to customers with an upfront discount.[565] In return for this upfront discount, customers agreed, through a shrink wrap, to use a cartridge only once and return the empty cartridge to Lexmark.[566] This meant a customer could not refill and reuse a Prebate cartridge.[567] "Non-Prebate" cartridges were sold without any upfront discount and were not subject to any reuse restrictions.[568] In addition to the contractual restrictions it imposed, Lexmark also used technical measures to restrict the type of cartridges used in its printers.[569] Each Lexmark toner cartridge contained a microchip that was used for a "secret handshake" with Lexmark printers.[570] The details of this secret handshake are reasonably complicated, but central to understanding the case.

Each Lexmark printer contained a program known as the Printer Engine Program.[571] The Printer Engine Program is a complex program used to control a variety of printer functions such as paper feed and movement and printer motor control.[572] Among the functions performed by the Printer Engine Program was verification of handshake signals sent to the printer by a cartridge being installed.[573] Each Lexmark toner cartridge contained a second program called the Toner Loading Program.[574] Unlike the Printer Engine Program, the Toner Loading Program was very simple.[575] One variant of the Toner Loading Program was 37 bytes in length, while the other variant was 55 bytes in length.[576]

562. Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522 (6th Cir. 2004).
563. *Id*. at 529.
564. *Id*.
565. *Id*.
566. *Id*. at 530.
567. *Id.*
568. *Lexmark*, 387 F.3d at 530.
569. *Id.*
570. *Id.*
571. *Id.*
572. *Id.*
573. *Id.*
574. *Lexmark*, 387 F.3d at 530.
575. *Id.*
576. *Id.*

The Toner Loading Program was used to measure the amount of toner remaining in a cartridge based on the amount of torque sensed on the toner cartridge wheel.[577] The Toner Loading Program was also used in the handshake process.[578] Each time a printer was turned on or when the printer door was opened, the Printer Engine Program would download a copy of the Toner Loading Program from the cartridge so toner levels in the cartridge could be measured.[579] As part of this downloading process, the Printer Engine Program performed a checksum[580] on the byte values of the Toner Loading Program.[581] The result of the checksum operation was compared to a value stored at a specific location in the printer cartridge.[582] If the values did not match, the Printer Engine Program assumed an error had occurred due to corruption of the Toner Loading Program.[583] If this error occurred, a message would be displayed to the user and the printer would not work until a printer cartridge was installed that was able to properly exchange handshake messages with the printer.[584] If the checksum value matched the value stored in the cartridge, the printer would operate properly.[585] Therefore, a Lexmark printer would only work if a cartridge was installed that downloaded a toner measuring program capable of producing the expected checksum.[586] This meant anyone wanting to create a Lexmark compatible cartridge needed to incorporate a verbatim copy of Lexmark's Toner Loading Program in their cartridge.[587] The defendant, Static Control Components ("SCC"), created microchips that contained such a verbatim copy.[588] The use of these microchips allowed other companies to create Lexmark-compatible toner cartridges.[589]

Lexmark sought to enjoin SCC's sales of its Lexmark-compatible microchips.[590] Lexmark alleged three theories of liability – two were based on the *Digital Millennium Copyright Act* (the "DMCA") and one was based on the *Copyright Act*.[591] The two actions under the DMCA

577. *Id.*
578. *Id.*
579. *Id.*
580. *Lexmark*, 387 F.3d at 531. The checksum was actually a secure hash created using the SHA-1 algorithm.
581. *Id.*
582. *Id.*
583. *Id.*
584. *Id.*
585. *Id.*
586. *Id.*
587. *Lexmark*, 387 F.3d at 531.
588. *Id.*
589. *Id.*
590. *Id.*
591. *Id.*

are not relevant to the topics under consideration. The *Copyright Act* action was a straightforward allegation that SCC's copying of the Toner Loading Program was an infringement of Lexmark's copyright in that program.[592]

Lexmark was successful in its motion for injunctive relief, and the district court ruled that Lexmark had established a likelihood of success in its copyright infringement claim.[593] The district court observed that the requisite level of creativity necessary to establish originality in a copyrighted work is extremely low and that the Toner Loading Program could have been written in multiple ways.[594] The district court rejected the various defenses asserted by SCC.[595] In particular, the district court ruled that the Toner Loading Program was not a lock-out program, and even if it was a lock-out program, security programs were just like any other computer program and were not inherently unprotectable.[596] The district court also rejected SCC's fair use and copyright misuse defenses.[597]

On appeal, the Sixth Circuit overruled the district court and remanded the case back to the district court for further proceedings.[598] However, the Sixth Circuit ruling was fragmented – producing a majority opinion, a concurrence, and a partial concurrence and partial dissent.[599] Unlike the cases discussed earlier, the *Lexmark* ruling is based primarily on merger. In the majority decision, the Sixth Circuit ruled the district court had committed three legal errors.[600] First, the district court had only applied the idea – expression dichotomy and accompanying principles of merger and scènes à faire in the second prong of the infringement test (substantial similarity) and not in the first prong of the infringement test (copyrightability).[601] In discussing this error, the Sixth Circuit observed that this is the approach taken in cases invoking Professor Nimmer's view that the idea-expression dichotomy "constitutes not so much a limitation on the copyrightability of works, as it is a measure of the degree of similarity that must exist between a copyrightable work and an unauthorized copy."[602] The Sixth Circuit stated that the copyrightability of a computer program does not turn solely on the availability of other options for writing the program and that such an ap-

592. *Id.*
593. *Lexmark*, 387 F.3d at 531.
594. *Id.*
595. *Id.*
596. *Id.* at 531-32.
597. *Id.*
598. *Id.* at 551.
599. *Lexmark*, 387 F.3d at 551.
600. *Id.* at 537.
601. *Id.* at 537.
602. *Id.* at 538.

proach would conflict with the Supreme Court's decision in *Feist*.[603] With respect to the issue of when to apply the idea–expression dichotomy and, in particular, when to apply the doctrines of merger and scènes à faire, the Sixth Circuit chose not to limit the use of the idea–expression dichotomy to the substantial similarity comparison.[604] In reaching this conclusion, the court observed that the idea–expression dichotomy is most commonly discussed during the consideration of substantial similarity because the copyrightability of a work is generally less frequently contested. The Sixth Circuit stated that the idea–expression dichotomy is not a measure of similarity, but instead a means to distinguish protectable elements from unprotectable elements.[605] Accordingly, the majority held that the idea–expression dichotomy is relevant during both phases of the infringement test since copyright protection extends only to expression and not to ideas.[606]

The second error committed by the district court was a failure to consider whether the Toner Loading Program could have been expressed in any other form when taking into consideration the functionality, compatibility and efficiency demanded of the program.[607] The Sixth Circuit considered the testimony of the party's respective experts regarding possible expression in the Toner Loading Program.[608] Based on the record before the court at the preliminary injunction phase, the Sixth Circuit concluded that the Toner Loading Program was not copyrightable.[609] However, the Sixth Circuit left the matter open for the district court to further examine this issue at the permanent injunction phase to determine (in accordance with the Sixth Circuit's ruling) whether the Toner Loading Program had sufficient originality to warrant copyright protection.[610]

Finally, and most significantly, the Sixth Circuit held that the district court erred in finding that the Toner Loading Program did not function as a lock-out code.[611] In this regard, the Sixth Circuit observed that if a single byte of the Toner Loading Program was changed, the resulting checksum value would change causing the handshake to fail with result that the printer would not operate.[612] Given these facts, the court concluded that the Toner Loading Program was a lock-out code because it

603. *Id.*
604. *Id.*
605. *Lexmark*, 387 F.3d at 538.
606. *Id.*
607. *Id.* at 539.
608. *Id.* at 538.
609. *Id.* at 541.
610. *Lexmark*, 387 F.3d at 541.
611. *Id.*
612. *Id.*

was necessary input to the checksum calculation and comparison.[613] Since the Toner Loading Program was a lock-out code for the printer, the merger and scènes à faire doctrines precluded it from copyright protection.[614] The majority opinion also addressed an issue raised in the dissent. In the dissenting opinion, Justice Feikens stated that the type of use made by the alleged infringer of the relevant subject matter was critical in determining whether merger had occurred. In response, the majority said:

> Judge Feikens is correct that a poem in the abstract could be copyrightable. But that does not mean that the poem receives copyright protection when it is used in the context of a lock-out code. Similarly, a computer program may be protectable in the abstract but not generally entitled to protection when used necessarily as a lock-out device.[615]

Ultimately, the court concluded, as did the courts in *Lotus*, *Bateman* and *Mitel*, that there was no copyright protection for the subject matter under consideration.[616] However, the majority did not dispose of the issue through a direct application of §102(b), but instead chose to deal with the issue through the merger and scènes à faire doctrines.

As mentioned earlier, the *Lexmark* decision has three separate opinions and a complete assessment of the case requires a consideration of all three opinions. In addition to the majority opinion, Justice Merritt wrote a concurring opinion, and Justice Feikens wrote a partially concurring and partially dissenting opinion. The concurring opinion by Justice Merritt deals with matters related to the *Digital Millennium Copyright Act*.[617] The concurring and dissenting opinion by Justice Feikens, however, contains a number of observations pertinent to the scope of protection for methods of operation.[618] In particular, Justice Feikens observed that there is some disagreement between various circuits about whether merger acts as a bar to copyrightability or simply as a defense to particular types of infringement.[619] He went on to observe that the Second and Ninth Circuits have taken the position that merger operates only as a defense to infringement while the Fifth Circuit has held that merger determines copyrightability.[620] Justice Feikens also noted that Professor Nimmer prefers the view that merger is a defense to infringement.[621] Finally, Justice Feikens stated it was his belief that the majority did not

613. *Id.*
614. *Id.* at 542.
615. *Id.* at 544.
616. *Lexmark*, 387 F.3d at 541.
617. *Id.*
618. *Id.* at *556*-57 (Feikens, J., concurring in part and dissenting in part).
619. *Id.*
620. *Id.* at 557.
621. *Id.* at 557 n.6.

take a position on this circuit split;[622] however, because of certain matters related to the various DMCA actions brought by the plaintiffs, Justice Feikens stated he believed that it was necessary to decide whether merger determines the initial question of copyrightability, or only operates as a defense to infringement.[623] This determination is also very relevant when assessing copyright issues related to dynamic linking and inter-process communication. In respect to this issue, Justice Feikens stated:

> I would exercise judicial economy and limit the holding to the case of merger with a method of operation (which is the question I believe this case presents), for the reasons below, I would find the merger doctrine can operate only as a defense to infringement in that context, and as such has no bearing on the question of copyrightability.[624]

---

622. *Lexmark*, 387 F.3d at 557 (Feikens, J., concurring in part and dissenting in part). This is an interesting observation. While it is true that the majority decision does not explicitly discuss the circuit split on merger, the majority seems to have been reasonably clear that merger is applicable during both phases of the infringement test. Hence, under the majority approach, it appears that merger can be used to determine copyrightability and as a defense to infringement. In particular the majority observed:

> In refusing to consider whether "external factors such as compatibility requirements, industry standards, and efficiency" circumscribed the number of forms that the Toner Loading Program could take, the district court believed that the idea-expression divide and accompanying principles of merger and scènes à faire play a role only in the "substantial similarity" analysis and do not apply when the first prong of the infringement test (copyrightability) is primarily at issue. *Lexmark*, 387 F.3d at 537-38.

Later the majority said:

> As a matter of practice, Nimmer is correct that courts most commonly discuss the idea-expression dichotomy in considering whether an original work and a partial copy of that work are "substantially similar" (as part of prong two of the infringement test), since the copyrightability of a work as a whole (prong one) is less frequently contested. But the idea-expression divide figures into the substantial similarity test not as a measure of "similarity"; it distinguishes the original work's protectable elements from its unprotectable ones, a distinction that allows courts to determine whether any of the former have been copied in substantial enough part to constitute infringement. Both prongs of the infringement test, in other words, consider "copyrightability," which at its heart turns on the principle that copyright protection extends to expression, not to ideas. *Id.* at 538.

The majority went on to cite, with approval, Mason v. Montgomery Data, Inc., 967 F.2d 135, 138 n.5 (5th Cir. 1992), rejecting the argument that the merger doctrine applies only to the question of infringement and noting that the Fifth Circuit has applied the merger doctrine to the question of copyrightability. *Id.* at 539.

623. *Id.* at 557 (Feikens, J., concurring in part and dissenting in part). Justice Feikens observed that the DMCA only protects works that are protected by Title 17 of the U.S. Code or in which the copyright owner has a right under Title 17. According to Justice Feikens if the doctrine of merger applied at the copyrightability stage, and merger was found to have occurred, then a plaintiff will have failed to state a claim upon which relief can be granted under the DMCA. However, if the merger doctrine is only applied at the infringement stage, then even if merger is found to have occurred, there will still be a requirement to determine whether protection might still be afforded under the DMCA.

624. *Id.* at 557 (footnotes omitted).

Expanding on this finding, Justice Feikens stated that:

> The Copyright Act denies copyright protection to, among other things, methods of operation. 17 U.S.C. § 102(b). However, an otherwise copyrightable text can be used as a method of operation of a computer-for instance, an original, copyrightable poem could be used as a password, or a computer program as a lock-out code. In my view, therefore, it is necessary to know what the potential infringer is doing with the material in order to know if merger has occurred. In other words, if I use my own copyrighted poem as a password or lock-out code, an individual who published the poem as part of a book could not escape a finding of liability for infringement. The rationale for the merger doctrine is that without it, certain ideas or methods of operation would be removed from the public realm because all ways of expressing them would be copyrighted. When a poem or program is used as a lock-out code, it is being used as a step in a method of operation of the thing it is locking. Therefore, to protect a work from copying when it is used as a password would be to prevent the public from using a method of operation.
>
> Under this reasoning, an individual who copied a poem solely to use as a password would not have infringed the copyright, because in that scenario, the alleged infringer would have the defense that the poem has "merged" with a method of operation (the password). By contrast, someone who copied the poem for expressive purposes (for instance, as part of a book of poetry) would not have this defense. For these reasons, I would hold that in cases where the merger is with a method of operation, the merger doctrine should be applied as a defense to infringement only, and not as informing the question of copyrightability of the work itself.[625]

The position taken by Justice Feikens is interesting and if other courts adopt this approach it could have significant implications for the use of dynamically linked libraries and inter-process communication. Under this approach, as long as any expressive material is used within a method of operation, a calling program should not become a derivative or infringing work of a called library because the substantial similarity test should not be satisfied due to merger with a method of operation. This is important since the viral provisions of the GPL are only engaged when another work is a derivative or infringing work of the GPL-licensed work.

The Feikens approach would also be significant for data structures used within a method of operation. In Justice Feikens' poem analogy, one could easily replace the poem with a data structure. Extending the analogy, if data structures are passed as parameters to a function or procedure (i.e. if they are part of a method of operation), then this type of use should not support a finding of substantial similarity. Even if the

625. *Id.* at 557-58.

data structures are inherently copyrightable, the fact they are being used as part of a method of operation should mean they merge with that method of operation and an alleged infringer should be able to use the merger defense to avoid a finding of substantial similarity.[626] If there is no substantial similarity, then the use of these data structures cannot make a calling program a derivative or infringing work of a linked library and the viral provisions of the GPL should not be engaged.

Accordingly, the scope of protection suggested in the dissenting opinion in *Lexmark* seems closer to *Lotus* than *Mitel*. Under the Feikens approach, expression that is potentially copyrightable is not protectable in the context of a method of operation.[627] This approach is similar to *Lotus*, which also does not protect expression necessary for the use of a method of operation. *Lotus*, however, does not deal with the treatment of that same expression outside the context of a method of operation. The Feikens approach contrasts with *Mitel*, which allows the protection of copyrightable expression within a method of operation.[628] Furthermore,

626. An interesting question is whether the use of such data structures in other parts of a program unrelated to the invocation of a GPL-licensed library will support a finding of infringement. Based on the Feikens logic, it would seem that these types of uses can potentially be infringing. In such scenarios, the use of such data structures will not be used in connection with a method of operation and hence merger should not apply since an inability to use those data structures in these circumstances will not prevent use of a method of operation. This situation is similar to the scenario described by Justice Feikens where someone copies a poem for expressive purposes, such as use as part of a book of poetry, which would not be entitled to the merger defense. Such an outcome raises difficult questions since there will always be debate about whether a particular use is sufficiently connected to a method of operation. For example, if a particular procedure performs data processing and returns a result through a data structure, can a calling program use that data structure in other parts of the calling program such as for further processing of the returned result? This question is certainly debatable; however, it seems reasonable to expect that some of the value in using a method of operation is attributable to an ability to subsequently use and process any results returned by that method of operation. Given the foregoing, it appears reasonable to conclude that the merger defense should not be limited to invocation and should be more broadly applied so a user of a method of operation can obtain the full benefit of that method of operation, including an ability to use and further process results obtained from that method of operation. However, in a case where a data structure used in a method of operation is used by another program for functions unrelated to that method of operation or exploitation of results returned by that method of operation, then a denial of the merger defense seems justified.

627. *Lexmark*, 387 F.3d at 556-57 (Feikens, J., concurring in part and dissenting in part).

628. After concluding that copyright protection for expressive parts of a method of operation was not denied as a matter of law, the *Mitel* court did not need to consider merger since the facts of the case allowed the Tenth Circuit to decide the matter because of a lack of originality. Accordingly, it is possible that if the Tenth Circuit had been considering a method of operation containing expressive subject matter it may have applied the merger doctrine and thus avoided infringement. That being said, the tenor of the decision suggests that the Tenth Circuit would not have taken that approach and would have found a suffi-

if the approach taken by Justice Feikens had been applied to *Bateman*, then the outcome probably would have been different than the actual case. Under the Feikens approach, the *Bateman* interface would almost certainly have been found to be a method of operation. Given the amount of material in the interface, it also seems likely a court would have found there was at least some copyrightable subject matter. A court probably would have also held that the copyrightable subject matter merged with a method of operation thereby preventing a finding of substantial similarity or copyright infringement because of the defense of merger with a method of operation. Therefore, the scope of protection for expression within a method of operation seems to be similar to the level of protection provided under *Lotus* and less than that provided under *Bateman* and *Mitel*.

x. Sega and Nintendo

There are two final cases of some relevance to the scope of copyright protection for methods of operation. Neither case deals with methods of operation in detail, but they are interesting because they contain fact situations similar to *Lexmark*. These cases are also interesting because one case implicitly provides protection to aspects of a method of operation while the other does not. The first case is *Atari Games Corp. v. Nintendo of America, Inc.*,[629] and the second case is *Sega Enterprises, Ltd. v. Accolade, Inc*.[630]These cases are similar to *Lexmark* because each deals with a lockout sequence for a game cartridge and console.[631] Both the *Lexmark* and *Sega* decisions purport to be consistent with the first of these cases, *Nintendo*, however, a closer examination of the ruling and facts in *Nintendo* suggests that *Sega* is not.

Nintendo considered various activities undertaken by Atari to try to develop game cartridges that were compatible with the Nintendo NES game console.[632] These activities included typical reverse engineering techniques such as electronic monitoring of communications between a Nintendo game cartridge and an NES console and chemical peeling of NES microchips.[633] An interesting aspect of the case is that Atari's reverse engineering efforts were unsuccessful until Atari obtained a copy of

---

cient amount of copyrightable subject matter within the method of operation to support a finding of substantial similarity and copyright infringement.

629. Atari Games Corp. v. Nintendo of America Inc., 975 F.2d 832 (Fed. Cir. 1992).

630. Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1992).

631. In Lexmark, the factual setting involved a toner cartridge that was plugged into a printer. The Sega and Nintendo cases dealt with game cartridges that were plugged into player consoles. In each case, the device receiving the cartridge checked for a particular data stream before it would allow the printer or game console to operate.

632. *Atari*, 975 F.2d at 836.

633. *Id.*

Nintendo's source code from the Copyright Office of the Library of Congress.[634] This copy of the Nintendo source code was in fact obtained in violation of Copyright Office rules.[635] After obtaining the Nintendo source code, Atari restarted its reverse engineering program and was eventually able to determine how the NES security system worked.[636] Once Atari understood the security system, it was able to create NES-compatible cartridges containing Atari games.[637] These Atari cartridges contained an Atari-developed unlocking program that was quite different from the one in the Nintendo cartridges.[638] However, both the Atari unlocking program and the Nintendo unlocking program produced the same data streams.[639] The Atari unlocking program was written for a different processor in a different programming language, but the Federal Circuit nonetheless found that Atari had copied enough protectable expression to support a finding of copyright infringement.[640] The court summarized its finding in the following passage:

> Finally, Nintendo seeks to protect the creative element of its program beyond the literal expression used to effect the unlocking process. The district court defined the unprotectable 10NES idea or process as the generation of a data stream to unlock a console. This court discerns no clear error in the district court's conclusion. The unique arrangement of computer program expression which generates that data stream does not merge with the process so long as alternate expressions are available. In this case, Nintendo has produced expert testimony showing a multitude of different ways to generate a data stream which unlocks the NES console.
>
> At this stage in the proceedings of this case, Nintendo has made a sufficient showing that its 10NES program contains protectable expression. After filtering unprotectable elements out of the 10NES program, this court finds no error in the district court's conclusion that 10NES contains protectable expression. Nintendo independently created the 10NES program and exercised creativity in the selection and arrangement of its instruction lines. The security function of the program ne-

634. *Id.*

635. *Id.* at 841-42.

636. *Id.* at 836.

637. *Id.*

638. *Atari*, 975 F.2d at 836. The Federal Circuit Court of Appeals described the Atari program as follows:

> The Rabbit uses a different microprocessor. The Rabbit chip, for instance, operates faster. Thus, to generate signals recognizable by the 10NES master chip, the Rabbit program must include pauses. Atari also programmed the Rabbit in a different language. Because Atari chose a different microprocessor and programming language, the line-by-line instructions of the 10NES and Rabbit programs vary.

639. *Id.* at 836 ("Atari's Rabbit program generates signals indistinguishable from the 10NES program"). *Id.* at 836-37 (stating that "the district court found, the Rabbit program generates signals functionally indistinguishable from the 10NES program").

640. *Id.*

> cessitated an original signal combination to act as a lock and key for the NES console. To generate an original signal, Nintendo had to design an original program. In sum, the district court properly discerned that the 10NES program contains protectable expression. At a minimum, Nintendo may protect under copyright the unique and creative arrangement of instructions in the 10NES program.[641]

This passage has been interpreted in subsequent cases to mean that merger does not occur if it is possible to write other programs that can effectively generate the unlocking sequence.[642] However, this appears to be exactly what Atari did. In describing the Atari unlocking program, the Federal Circuit noted that it was written on a different processor using a different programming language.[643] In fact, since the Atari processor was faster than the Nintendo processor, the Atari program had to include pause instructions so the data stream it produced would match the unlocking sequence expected by a Nintendo console.[644]Consequently, the most significant similarity between the two programs was that they produced the same data stream. Nonetheless, the Federal Circuit ruled that:

---

641. *Id.* at 840 (citations omitted).

642. *For example*, in Lexmark Int'l., Inc. v. Static Control Components, Inc., 387 F.3d 522, 543 (6th Cir. 2004), the court observed:

> The Federal Circuit's rationale for accepting copyright protection for the 10NES program does not undermine our conclusion because the 10NES program was not a "lock out" code in the same sense that the Toner Loading Program is. In *Atari*, the data bytes of the 10NES program did not themselves do the "unlocking" of the game console; *the program, when executed, generated an arbitrary stream of data that in turn enabled the console to function. That same data stream, the court concluded, could have been produced by a number of alternate programs; for this reason, the expression contained in the computer program did not "merge" with the process.* ("The unique arrangement of computer program expression which generates that data stream does not merge with the process so long as alternative expressions are available. In this case, Nintendo has produced expert testimony showing a multitude of different ways to generate a data stream which unlocks the NES console.") (citations omitted) (emphasis added).

In Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510, 1524 n.7 (9th Cir. 1992), the court observed:

> We therefore reject Sega's belated suggestion that Accolade's incorporation of the code which "unlocks" the Genesis III console is not a fair use. Our decision on this point is entirely consistent with *Atari v. Nintendo*, 975 F.2d 832 (Fed. Cir. 1992). Although *Nintendo* extended copyright protection to Nintendo's 10NES security system, that system consisted of an *original program* which generates an arbitrary data stream "key" which unlocks the NES console. Creativity and originality went into the design of that program. *See id.* at 840. Moreover, the federal circuit concluded that there is a "multitude of different ways to generate a data stream which unlocks the NES console." *Atari*, 975 F.2d at 839. The circumstances are clearly different here. Sega's key appears to be functional. It consists merely of 20 bytes of initialization code plus the letters S-E-G-A. There is no showing that there is a multitude of different ways to unlock the Genesis III console.

643. *Atari*, 975 F.2d at 840.

644. *Id.*

> Nintendo's 10NES program contains more than an idea or expression necessarily incident to an idea. Nintendo incorporated within the 10NES program creative organization and sequencing unnecessary to the lock and key function. Nintendo chose arbitrary programming instructions and arranged them in a unique sequence to create a purely arbitrary data stream. This data stream serves as the key to unlock the NES. Nintendo may protect this creative element of the 10NES under copyright. External factors did not dictate the design of the 10NES program. Nintendo may have incorporated some minimal portions of the program to accommodate the microprocessor in the NES, but no external factor dictated the bulk of the program. Nor did Nintendo take this program from the public domain.[645]

In its analysis the Federal Circuit agreed with the district court that the unprotectable idea or process is the generation of a data stream to unlock a console.[646] Both the district court and the Federal Circuit Court agreed that the unique sequence of instructions used to create the arbitrary data stream is protectable.[647] Given that Atari wrote a different program to create the arbitrary data stream used by Nintendo, the only logical conclusion is that Ninth Circuit's ruling entails protection of the arbitrary data stream. The court supported its finding by referring to expert testimony that there were "a multitude of different ways to generate *a data stream* which unlocks the NES console." (emphasis added). This statement is correct in the sense that theoretically there are a large number of data streams that will work as an unlocking protocol provided both the sender and the receiver have agreed on the particular data stream to be used as the key. However, if a console is expecting only one particular data stream, or if the console is expecting a data stream with certain characteristics, then only that particular data stream or those data streams having those expected characteristics will function correctly. In such a situation, the data steam is a part of the method of operation or merges with the method of operation for the cartridge/console system and the data stream should not be protected under copyright. Neither the Federal Circuit nor the district court in this case seems to have considered whether the NES unlocking system was a method of operation, or whether the NES data stream was part of or merged with a method of operation. As a result, those courts appear to have granted *de facto* copyright protection to the method of operation for unlocking the Nintendo NES console. Although *Sega* claims to be in accord with *Nintendo*, *Sega* takes a very different approach and has a very different outcome. *Lexmark*, however, actually is in accord with *Nintendo*, since *Lexmark* is a case where no other program could have

645. *Id.*

646. *Id.*

647. *Id.*

been written that would unlock a Lexmark printer. In *Sega*, the decision was more focused on Accolade's reverse engineering activities and the fair use arguments of the respective parties, however, a necessary ruling in *Sega* was that the unlocking sequence was not protected by copyright.[648] In *Lexmark*, both the majority and dissenting opinions found that merger occurred and that copyright protection should not be extended to the expression embodied in the unlocking mechanism.[649]

For these reasons neither *Sega* nor *Nintendo* provide any significant guidance in understanding the scope of copyright protection for methods of operation. In *Sega*, the focus is on fair use as a defense to allegations of copyright infringement in connection with reverse engineering activities. The issue of protection for methods of operation was secondary and, as such, did not receive significant attention from the court. In *Nintendo*, the court seems to have missed the fact that the unlocking system was a method of operation. The court also seems to have misunderstood the testimony of one of Nintendo's expert witnesses. Accordingly, the decision that the NES unlocking key and protocol should have been protected is probably incorrect.

4. *GPL "Viral" Effects and Dynamic Linking*

This section will analyze whether a program being dynamically linked to a GPL-licensed library is a derivative or infringing work of that library and whether the viral provisions of the GPL are engaged. As described in Section II.C, a dynamically linked ELF object file has a symbol table containing the symbol names used by that program. Among these symbol names will be any external symbols from any GPL-licensed libraries linked to the program. These symbols are components of the API of those the GPL-licensed libraries. Unlike a statically linked program, a dynamically linked program will not contain any object code from any linked GPL-licensed libraries. However, the object code of a dynamically linked program may contain non-literal elements that have been copied from linked GPL-licensed libraries.

As with static linking, the derivative works analysis begins with a consideration of the transformative requirement. Similar to static link-

---

648. Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510, 1552 (9th Cir. 1992). Unfortunately, other than observing that Accolade copied Sega's software solely to discover the functional requirements for compatibility and that the Sega key appeared to be functional, the court did not provide a specific rationale for denying copyright protection to the unlocking key.

649. The majority held that the Toner Loading Program was not copyrightable, Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 544 (6th Cir. 2004). The dissent would have held that there was no copyright infringement because of the defense of merger with a method of operation. *Id.* at 557 (Feikens, J., concurring in part and dissenting in part).

ing, the APIs or technical interfaces for an object library are in a form that allows portions of those APIs or interfaces to be included in other programs. The actual use of parts of those APIs or interfaces in a dynamically linked object code program will allow one program to pass execution from its own object code to the object code of the called library. Accordingly, it seems reasonable to conclude that the transformative requirement is met for dynamic linking because the copied material has been adapted to allow it to fulfill a different objective. In one case, the copied material is in a state that allows for its incorporation into other programs for the creation of executable code. In the other case, the copied material is actually present in an executing program and it allows control to be passed back and forth between different executing processes.

Once the copied material has been identified and once it has been determined that the transformative requirement has been met, the final step in the derivative works analysis is to determine whether the copied material represents a substantial copying of the pre-existing work. In the case of technical interfaces and data structures, the analysis becomes more difficult. First, the proper treatment of technical interfaces under copyright law needs to be determined. To date, most cases have treated technical interfaces as methods of operation without conducting any significant analysis. In the one case that did consider the meaning of the term "method of operation," this term was given a very broad construction.[650] Applying the construction given to this term in *Lotus*, it would seem that the API of a dynamically linked library qualifies as a method of operation. The API is the means by which a programmer operates or accesses the underlying functionality contained in a dynamically linked library.[651] Additionally, without an API, a programmer would not be

650. Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807, 815 (1st Cir. 1995). The First Circuit Court of Appeals stated that it interpreted the term "method of operation" as used in §102(b) to refer to "the means by which a person operates something, whether it be a car, a food processor, or a computer." *Id.* The court also observed that the Lotus command hierarchy provided the means by which users control and operate Lotus 1-2-3. *Id.* Further, the court observed that without the menu command hierarchy, users would not be able to access and control or make use of the Lotus 1-2-3 functional capabilities. *Id.*

651. There is one difference between the Lotus fact setting and the hypothetical setting for a dynamically linked library. Lotus dealt with human end users interacting with an application program. For dynamically linked libraries the situation is different because one program will be calling another program without direct end user action. However, it appears this difference does not have any legal significance. First, there is no indication that the Lotus court intended to limit its interpretation of the term method of operation to situations involving human action. The language used by the Lotus court is probably a function of the facts in that case (i.e. a human – computer interface). Second, if human participation was a requirement, then the actions of computer programmers who choose to invoke and use an API could potentially fulfill this requirement. In such a scenario, the API programmers can be viewed as making a method of operation available for invocation,

able to operate or access the functionality contained within a dynamically linked library. Furthermore, the main purpose of an API is to allow programmers and other programs to operate or use the capabilities contained within a particular dynamically linked library. Accordingly, it seems reasonable to conclude that under the criteria set forth in *Lotus*, an API or technical interface should qualify as a method of operation.

An examination of the dictionary meaning of the constituent words of "method of operation" leads to a similar conclusion. The Merriam-Webster Online Dictionary defines "method" as "a way, technique, or process of or for doing something"; "a procedure or process for attaining an object"; or "a systematic procedure, technique, or mode of inquiry employed by or proper to a particular discipline or art."[652] The same dictionary defines "operation" as "a method or manner of functioning"; "performance of a practical work or of something involving the practical application of principles or processes"; or "any of various mathematical or logical processes (as addition) of deriving one entity from others according to a rule."[653] Other English and legal dictionaries have similar definitions.[654] An API has been variously defined as: "[a] set of standards or conventions by which programs can call specific operating system or network services";[655] "[a] functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program";[656] "[a]n interface that is defined in terms of a set of functions and procedures, and enables a program to gain access to facilities within an application";[657] and "[a] language and message format used by an ap-

---

and the programmers who use an API within their programs actually select when that method of operation is to be invoked (although the invocation typically occurs well after development when a calling program is actually running). In this regard, the invocation of an API is similar to the invocation of the macro reader facility in Lotus 1-2-3. In both cases, the designers of the method of operation established a framework under which the method of operation can be subsequently invoked in a machine-to-machine setting without direct human action. Given that the First Circuit Court of Appeals considered the macro reader facility and chose not to treat it any differently than the human – computer portions of the Lotus 1-2-3 menu command hierarchy strongly suggests that the method of operation test established in that case was not meant to be restricted to situations involving direct human actions.

652. Merriam Webster Online Dictionary, Method, http://www.m-w.com/dictionary/method (last visited Apr. 26, 2009).

653. Merriam Webster Online Dictionary, Operation, http://www.m-w.com/dictionary/operation (last visited Apr. 26, 2009).

654. *See*, Stacey H. King, *Are We Ready to Answer the Question?: Baker v. Selden, The Post-Feist Era, and Database Protections*, 41 J.L. & TECH. 65 (2001).

655. WEBSTER'S NEW WORLD COMPUTER DICTIONARY 24 (9th ed. 2003).

656. IBM DICTIONARY OF COMPUTING 28 (10th ed. 1994).

657. A DICTIONARY OF COMPUTING 19 (Oxford University Press 5th ed. 2004).

plication program to communicate with another program that provides services for it."[658]In *United States v. Microsoft Corp.*, an API was characterized as "synapses at which the developer of an application can connect to invoke pre-fabricated blocks of code in the operating system. These blocks of code in turn perform crucial tasks, such as displaying text on the computer screen."[659] As can be seen from these definitions, an API is a specific, structured way of accessing or invoking software capabilities. Based on these definitions, it is difficult to characterize an API as anything other than a "way, technique or process for doing something," "a systematic procedure, technique, or mode of inquiry employed by or proper to a particular discipline or art,"[660] or "any of various mathematical or logical processes (as addition) of deriving one entity from others according to a rule."[661] Hence, an examination of the dictionary meaning of the constituent words of "method of operation" and typical industry definitions for application programming interfaces suggests that APIs are methods of operation.

As demonstrated in the case law review, there seem to be at least three distinct approaches to determining the scope of copyright protection for interfaces/methods of operation. The first approach is that taken by the First Circuit in *Lotus v. Borland*, in which that court held that copyright protection is denied to any copyrightable expression that is a necessary part of a method of operation. The second approach is that taken by the Tenth Circuit in *Mitel v. Iqtel,* in which that court would grant copyright protection to certain subject matter notwithstanding the fact that such subject matter may be embodied within a method of operation at a higher level of abstraction. Finally, there is the approach taken by the Sixth Circuit in *Lexmark v. Static Control*, in which the majority applied the merger and scènes à faire doctrines to determine issues of copyrightability as well as substantial similarity, while the minority would only have applied these doctrines as a defense during the substantial similarity analysis. Under the majority approach in *Lexmark,* if certain expression merges with a method of operation, then copyright will not subsist in that expression. Under the minority approach, copyright can subsist in expression that merges with a method of operation (provided the originality requirement is met), but merger may be used as a defense against an allegation of infringement when the expression is used in the context of the method of operation.

The scope of protection for interfaces/methods of operation has also been considered by the Eleventh Circuit. However, the Eleventh Circuit,

658. ALAN FREEDMAN, THE COMPUTER GLOSSARY: THE COMPLETE ILLUSTRATED DICTIONARY 11 (9th ed. 2001).

659. United States v. Microsoft Corp., 84 F. Supp. 2d 9, 12 (D.D.C 2000).

660. Merriam Webster Online Dictionary, Method, *supra* note 652.

661. Merriam Webster Online Dictionary, Operation, *supra* note 653.

while tending towards a *Mitel*-like scope of protection, does not appear to have adopted a definitive position.

Given that there seems to be three different approaches for determining the scope of protection for expression embodied within a method of operation, the next question is whether these approaches can lead to different results when applied to dynamic linking. The answer to this question appears to be yes. Under the *Lotus v. Borland* approach the fact that expression may be embodied in a method of operation is immaterial. The initial inquiry to be made by a court is whether a particular technical interface is a "method of operation." If so, then no further inquiry is required because under §102(b) any expression embodied in a method of operation that is necessary for its use is denied copyright protection. Accordingly, under the *Lotus* approach the use of an application programming interface from a GPL-licensed library should not engage the viral provisions of the GPL. As discussed earlier, the viral provisions of the GPL should not be engaged unless a program uses material from a GPL-licensed library sufficient to make that program a derivative or infringing work of the library. If unprotected subject matter is used, that subject matter cannot make a linking program a derivative or infringing work of a library it is calling.

A subsequent work will become a derivative or infringing work of a pre-existing work if that subsequent work has copied sufficient protectable expression from the pre-existing work to make it substantially similar to the pre-existing work. This comparison is qualitative rather than quantitative, and the copying of a small but significant portion of a pre-existing work can support a finding of substantial similarity. A key point about the substantial similarity test is that it is based only on the copying of protectable subject matter. When a calling program uses an API to access functionality provided by a GPL-licensed library, the source code of that program will contain the symbols, data structures, function calls and procedure calls needed to allow the applicable compilers and linkers to create object code files that can pass control back and forth at run-time. The object code version of the calling program will similarly contain the necessary symbols and information to allow the operating system to dynamically link the calling program to the called GPL-licensed library so that execution can be passed back and forth between the calling program and the library. Under *Lotus*, an API is almost certainly a method of operation. Once an API is classified as a method of operation, the analysis is very straightforward – copyright protection is denied under §102(b) to those portions of the API that are necessary for its use, and that subject matter cannot be used as a basis for a finding of substantial similarity.

However, the exclusion of expression contained within an API used to call a dynamically linked library does not conclude the substantial

similarity analysis. It is still possible that other material may have been copied from a linked GPL-licensed library that can support a finding of substantial similarity. Unlike static linking, none of the code from a dynamically linked GPL-licensed library is actually copied into a calling program. Nonetheless, the substantial similarity test can still be satisfied through copying of non-literal elements of a dynamically linked library. The most likely source of such non-literal expression would be the data structures used in a GPL-licensed library that must also be used in a calling program. *Baystate* suggests that the expressive choices made in selecting data structure names and arrangements are not copyrightable. This conflicts with the findings in *Positive Software*, a case that seems to have been decided on much sounder copyright principles. There are also numerous other cases that have held that data structures can be protected as non-literal elements of a computer program.

Therefore, it seems likely that the copying of data structures from a GPL-licensed library can support a finding of substantial similarity – provided that such copying meets the requisite qualitative threshold and further provided that copyright protection for those data structures is not denied because of any limiting doctrines. When a program calls a GPL-licensed dynamically linked library, the first potentially relevant limiting doctrine is §102(b). If a data structure is part of an API and hence part of the method of operation for a dynamically linked library, then under a *Lotus*-like analysis any expression embodied in that data structure that is necessary for the use of the API should not be protected by copyright because to do so would mean that the method of operation could not be used without infringing that copyright. The use of data structures within an API is extremely common and arises most frequently when a parameter in a function call is specified to be a particular data structure. Under a *Lotus*-like approach the use of potentially expressive data structures within an API for the purpose of invoking a method of operation should not give rise to copyright infringement. However, if a data structure is used within an API, it is common for that data structure to be used in other parts of a calling program to allow the calling program to obtain the full intended benefit of the capabilities provided through the API. This raises a question about how these other uses should be treated. This question did not arise in *Lotus* and hence was not considered by that court. However, it is possible to examine the principles enunciated in *Lotus* to predict how this issue might have been resolved.

The first question to consider is whether §102(b) precludes protection of expression used within a method of operation when that expression is used in other parts of a calling program for other purposes. The answer to this question appears to be no. The rationale for §102(b) and for its application in *Lotus* was to prevent copyright from being used to

provide a *de facto* monopoly over ideas and over the various manifestations of ideas such as procedures, processes, systems and methods of operation. If the expressive parts of a method of operation are used elsewhere in a calling program, these uses will need to be examined to determine whether copyright protection in these instances would pose the same threat as that addressed by §102(b). If these other uses cannot create a *de facto* monopoly over a method of operation or some other category covered by §102(b), then the underlying rationale for the application of §102(b) will not exist. Therefore it would appear that copyright protection is available for expression used within a method of operation when that expression is used in other parts of a program for purposes unrelated to the method of operation. That being said, a court applying this approach should give reasonably wide latitude for uses that are related to the use of a method of operation. For example, if a particular procedure is intended to return a result via a parameter that is a complex data structure, it is reasonable to assume that the calling program will utilize that result in other contexts. In such a case, a calling program might display the result to a user, might perform some subsequent processing on the result, or might store the result in a data file to allow a user to subsequently view the result or perform additional processing. Merely allowing a method of operation to be invoked is of little use unless other actions and/or operations that flow naturally from the use of that method of operation are also permitted. Accordingly, if the purpose of §102(b) is to be fulfilled, these related uses should not support a finding of copyright infringement. If copyright protection is provided in these situations, then the effect will be to provide the *de facto* monopoly that §102(b) sought to avoid because only the developer of a method of operation will be able to make full subsequent use of its functionality. Accordingly, §102(b) should apply to actions or operations that are reasonably connected to the use of the functionality provided by a method of operation. However, while reasonable latitude should be given to allow full exploitation of a method of operation, uses that are not reasonably related to the use of that method of operation should still support a finding of copyright infringement.

If §102(b) is not applicable, then the next issue to consider is originality. There are two aspects of originality that will need to be considered. The first is whether a data structure meets the copyright threshold for originality. As articulated in *Feist*, this standard is very low and requires only that there is some "creative spark." In most cases this very low threshold will be met. However, there will undoubtedly be some instances where a very minimal data structure will not meet this thresh-

old.[662] From a practical perspective, the more interesting aspect of originality is the requirement for the subject matter not to have been copied from another source. Often a requirement or desire for compatibility with other software will dictate the use of data structures from other sources such as public standards or other non-GPL licensed software.[663] In these cases, the copied data structures will not be original to the GPL-licensed library using them and this material cannot be used to establish substantial similarity or infringement. This result is very important because there are literally hundreds if not thousands of packages built on data structures developed in conjunction with the UNIX operating system and the various communications protocols supported by that operating system.

Therefore under *Lotus* the use of third-party defined data structures combined with the ability to use APIs (methods of operation) means that theoretically there should be a reasonably broad ability to dynamically link to and interoperate with GPL-licensed libraries without engaging the viral provisions of the GPL. However, because *Lotus* is currently only binding in the First Circuit, and because any successful commercial software product will be licensed throughout the entire United States, the jurisprudence in other circuits must be examined to determine whether dynamic linking to a GPL-licensed library involves significant risk.

The next circuit to consider is the Sixth Circuit and the analytical approach set forth in *Lexmark v. Static Control*. The key opinions in this fragmented decision are the majority opinion written by Justice Sutton and the partially dissenting, partially concurring opinion written by Justice Feikens. Both opinions are worth considering since each of them discusses matters relevant to the scope of protection for methods of operation.

The majority and minority opinions in *Lexmark* each use the merger doctrine and, to a lesser extent, the scènes à faire doctrine.[664] It is easier to consider the minority opinion first since the reasoning in that opinion is clearer. Justice Feikens observed that there is a circuit split about whether merger acts as a bar to copyrightability or whether it is simply a

662. For example, a data structure that mirrored the telephone white pages listings considered in *Feist* in which each individual entry contained a name, address and telephone number, would probably not have the necessary creative spark for copyright protection.

663. For example, GPL-licensed software designed to provide email functionality may use data structures copied from one or more RFCs that specify various email standards such as Simple Mail Transfer Protocol (RFC 821) or Post Office Protocol – Version 3 (RFC 1939).

664. Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522 (6th Cir. 2004).

defense to particular types of infringement.[665] Justice Feikens also observed that the choice of one approach over the other is important and illustrated this point through an example involving a copyrightable poem.[666] In one part of the example, the poem is used as a password or lockout code and is copied solely for those purposes.[667] In the other part of the example, the same poem is copied and used as part of a book of poetry.[668] Justice Feikens states that when the poem is used as a password or lockout code (i.e. as a necessary part of a method of operation), the poem should merge with the method of operation and the merger defense should be available to anyone who copies the poem for that use.[669] When the same poem is copied and used in a book of poetry, even though the poem was originally created and used as a lockout code, Justice Feikens states that the defense of merger should not be available.[670] This result can only be achieved if merger is applied as a defense to infringement and not in an initial determination about copyrightability. If merger is used to determine copyrightability, then copyright will be denied in all circumstances and the use of the copied poem in a book of poetry will not be infringing. Under the Feikens approach, an expressive part of a method of operation may not be protected by copyright when used in the context of a method of operation, but if that expression is used in other contexts then it may be protected.

The approach taken by the majority in *Lexmark* is not as clearly articulated as that of Justice Feikens. In the majority opinion, Justice Sutton ruled that the district court had erred in three ways.[671] First, the district court determined copyrightability incorrectly by simply examining whether a particular work could have been put together in a number of different ways.[672] Justice Sutton stated that a court must examine alternative ways of putting a work together and determine whether they are feasible in the given setting.[673] In particular, Justice Sutton stated that the idea-expression divide and the accompanying principles of merger and scènes à faire inform both the substantial similarity test and the copyrightability test.[674] The second error that Justice Sutton identified flowed from the first error. Given the mistaken approach taken by the district court, the majority found that the constraints on the Toner

665. *Id.* at 556-57 (Feikens, J. concurring in part and dissenting in part).
666. *Id.* at 557.
667. *Id.* at 558.
668. *Id.*
669. *Id.*
670. *Lexmark Int'l*, 387 F.3d 522 at 558 (Feikens, J. concurring in part and dissenting in part).
671. *Lexmark Int'l,* 387 F.3d at 537-42.
672. *Id.* at 537.
673. *Id.* at 537-38.
674. *Id.*

Loading Program needed to be reconsidered.[675] Finally, the majority held that the district court erred in assessing whether the Toner Loading Program functioned as a lockout code.[676] This error is the most significant one when considering copyright protection for methods of operation.[677] Unfortunately, the majority did not definitively state whether the Toner Loading Program was uncopyrightable because merger with a method of operation acts as a bar to copyrightability or because merger is a defense to an allegation of copyright infringement.[678]This lack of clarity is frustrating given the statements made by the majority earlier in the decision about merger and scènes à faire being relevant at both the copyrightability phase and the substantial similarity phase of the infringement test.[679] The failure by the majority to definitively state its position on this issue does not affect an analysis of the use of expressive parts of an API for calling dynamically linked functions. However, it does affect an analysis of the use of expressive parts of an API in other circumstances.

The approach taken by the Sixth Circuit appears to provide a level of protection similar to that under *Lotus*. Under the minority approach, the use of potentially copyrightable expression in a method of operation will not support a finding of substantial similarity or infringement because the defense of merger with a method of operation is available. Since there can be no substantial similarity, a calling program cannot become a derivative or infringing work of a GPL-licensed library simply by invoking a procedure defined in its API. However, while Justice Feikens was very clear that a copyrightable expression that merges with a method of operation will not be protected in that context, if that expression is used in other contexts it can be protected. Accordingly, if expressive parts of the API of a GPL-licensed library are used in other parts of a calling program in contexts unrelated to the use of the API, those uses can potentially support a finding of substantial similarity and infringement. This is possible under the Feikens approach because merger does

675. *Id.* at 539.

676. *Id.* at 541.

677. *Lexmark Int'l*, 387 F.3d at 541.

678. *Id.* at 542. On this issue, the majority cryptically said, "[o]n this record, pure compatibility requirements justified SCC's copying of the Toner Loading Program." *Id.*

679. *Id.* at 544. The majority made this final statement:

> For like reasons, Judge Feikens is correct that a poem in the abstract could be copyrightable. But that does not mean that the poem receives copyright protection when it is used in the context of a lock-out code. Similarly, a computer program may be protectable in the abstract but not generally entitled to protection when used necessarily as a lock-out device. *Id.*

The phrase "not generally entitled to protection" in this passage is ambiguous and it is not clear whether this means the program is not copyrightable or that in certain circumstances merger or some other doctrine may be available as a defense to an allegation of copyright infringement.

not extinguish copyright. Therefore, use of such expression in other parts of a program not related to the use of the API can support a finding of substantial similarity because the merger defense will not apply. In these situations, the viral provisions of the GPL can be engaged.

As discussed above in the *Lotus* analysis, expression within an API will generally need to be used in other parts of a calling program to fully exploit the capabilities provided by the API. Such uses should not be viewed as uses in another context, and, for the purpose of a Feikens-like analysis, the defense of merger with a method of operation should still be available to permit full exploitation of the capabilities provided by the API.[680] The merger defense should only be denied for uses that are completely unrelated to the exploitation of the API being invoked.

Therefore the approach taken by the minority in *Lexmark* is similar to *Lotus* in that expression embodied in a method of operation cannot be used to support a finding of substantial similarity and infringement. In *Lotus*, the court used §102(b); in *Lexmark*, Justice Feikens used the defense of merger with a method of operation. The Feikens approach differs from *Mitel* because the Tenth Circuit will protect copyrightable expression notwithstanding the fact that such expression may be embodied in a method of operation at a higher level of abstraction.[681] The minority approach in *Lexmark* and the *Mitel* approach are similar in that both allow copyright to subsist in the expressive aspects of a method of operation. However, in *Lexmark*, the availability of merger means that copyright will not be enforced to the extent any expression is needed for

680. *Id.* at 558 (Feikens, J. concurring in part and dissenting in part). Justice Feikens said of the merger defense "Defendant can only claim this defense to infringement if it uses the TLP to interface with the Lexmark printers at issue, and if it is a necessary method of operation of the machine." *Id.* Given this language, it is not clear whether Justice Feikens would allow the merger defense for ancillary uses of protectable expression within an API. (It is also unclear how Justice Feikens would handle a situation involving a program with two methods of operation – as was the case in Lotus. If there are two methods of operation for a program, then it is possible that neither will be deemed "necessary" because of the existence of the other.) There is no doubt that use of an API is necessary for the use of a dynamically linked library (there is no other way to programmatically invoke the library). However, a question arises about what is meant by using the expression to interface with the library. This requirement could be read narrowly to mean only those uses that are strictly required to invoke the functionality of the library or it could be read broadly to mean the ability to fully utilize the functionality and results provided by the library. This issue is particularly important for data structures that are an inherent component of an API and that are used to transmit and receive data between a calling program and a called library. If full use of a given method of operation is to be allowed, then it appears that use of parts of an API beyond simple invocation will need to be permitted.

681. Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1372 (10th Cir. 1997).

the use of a method of operation.[682] Finally, both the minority in *Lexmark* and the court in *Mitel* will protect expression embodied in a method of operation when that expression is used in other contexts.[683]

The scope of protection the majority in *Lexmark* accords to the API of a GPL-licensed library is similar to that provided by the minority. However, because the majority did not state definitively whether merger is a bar to copyright or simply a defense to a claim of copyright infringement, the exact bounds of the protection the majority would provide are not as clear as that for the minority. Like the Feikens approach, the majority would not allow expression embodied in a method of operation to be used to support a finding of substantial similarity. Accordingly, under the majority approach a calling program will not become a derivative or infringing work of a GPL-licensed library simply by invoking a function within its API and consequently the viral provisions of the GPL should not be engaged. However, since the majority did not explicitly state whether merger is a defense to a claim of copyright infringement or a bar to copyrightability, it is not clear whether use of these parts of an API in other parts of a calling program will support a finding of substantial similarity. If the majority intended merger with a method of operation to serve as a bar to copyright, then the use of expressive portions of an API in other contexts within a calling program will not engage the viral provisions of the GPL. If the majority did not intend merger to act as a bar to copyrightability, but instead as a defense to an allegation of copyright infringement, then the use of such expression in other contexts can engage the viral provisions of the GPL. In this latter case, the scope of protection accorded to expression embodied in a method of operation is the same as that provided by the minority.

The final circuit-specific approach to the scope of protection for methods of operation is that established by the Tenth Circuit whose analytic approach is set forth in *Mitel v. Iqtel*. Of the three main positions on the scope of protection for methods of operation, *Mitel* is perhaps the easiest to analyze. Unfortunately, in practice, *Mitel* is probably the most difficult to apply.

As discussed earlier, the Tenth Circuit rejected the approach taken by the First Circuit in *Lotus*.[684] The Tenth Circuit did not agree with the First Circuit's decision not to use the Abstraction-Filtration-Compar-

682. When invoking an API, it appears that the merger defense will almost always be available because the way in which various functions are made available through an API is strictly specified and these functions will not work unless these conventions are followed.

683. In *Mitel*, this result flows naturally from the court's willingness to protect expression even when it is embodied in a method of operation at a higher level of abstraction. *See Mitel*, 124 F.3d at 1372.

684. *Id.*

ison method as an analytical framework.[685] Furthermore, the Tenth Circuit did not agree with the First Circuit's conclusion that copyright protection for otherwise protectable expression embodied in a method of operation at a higher level of abstraction is excluded under §102(b).[686] Instead, the Tenth Circuit chose to use the Abstraction-Filtration-Comparison method, and further concluded that §102(b) does not extinguish the protection accorded a particular expression of an idea merely because that expression is embodied in a method of operation at a higher level of abstraction.[687]

The Tenth Circuit's approach allows expression within a method of operation to be protected under copyright. Therefore, if a calling program invokes the expressive portions of an API, then the use of that expression may make the calling program substantially similar to the called library. Therefore, a program calling a GPL-licensed library may become a derivative or infringing work of that library and thereby engage the viral provisions of the GPL.

As a practical matter, the *Mitel* test will be difficult for commercial software developers to use. There are a number of inherent uncertainties in the test that make it unlikely to produce definitive answers. The first aspect of the *Mitel* test that makes it difficult to apply is the use of the Abstraction-Filtration-Comparison test as an analytical framework. The Abstraction phase is particularly important because it establishes the general framework for separating idea from expression. In the context of the *Mitel* test, the Abstraction phase takes on added importance because the Tenth Circuit is willing to protect expression that may be embodied in a method of operation at a higher level of abstraction.

Unfortunately, the Abstraction phase of the Abstraction-Filtration-Comparison test is the least well defined, and, as a result, is the most difficult to apply and the most prone to disagreement.[688] Many courts applying the Abstraction-Filtration-Comparison test have either skipped the Abstraction phase or provided virtually no analysis for the chosen level of abstraction.[689] As a result, it will be very difficult for commercial software developers to determine whether a method of operation is unavailable because it is at "too high" a level of abstraction.

The difficulty with the *Mitel* test is that once it becomes possible to protect expression within a method of operation it becomes very difficult

685. *Id.*

686. *Id.*

687. *Id.*

688. Peter Pan Fabrics, Inc. v. Martin Weiner Corp., 274 F.2d 487, 489 (2nd Cir. 1960) ("Obviously, no principle can be stated as to when an imitator has gone beyond copying the 'idea', and has borrowed its 'expression'. Decisions must therefore inevitably be *ad hoc*").

689. *See* Bateman v. Mnemonics, Inc., 79 F.3d 1532 (11th Cir. 1996); and *Mitel*, 124 F.3d 1366 (10th Cir. 1997).

for developers to identify expression that may attract this protection. The combination of the *Mitel* test with the low originality threshold for copyright protection will mean that most methods of operation will contain at least some protectable expression. Under *Mitel* it is also more likely that the use of data structures embodied in an API will support a finding of substantial similarity. If the *Mitel* approach is widely adopted it will still be possible to dynamically link to a library without making the calling program a derivative or infringing work of that library. However, given the risks associated with the viral provisions of the GPL and the difficulty in applying the *Mitel* test, if *Mitel* is generally adopted it seems unlikely that many companies will take the risk of dynamically linking their commercial programs to GPL-licensed libraries.

A potential exception may occur for GPL-licensed libraries implementing third-party standards. In this special case, the API and data structures and protocols will not be original to the GPL-licensed library and the author of the library will have to have copied them from elsewhere. In this situation, even under *Mitel*, invocation of this API and use of the accompanying data structures and protocols cannot make a calling program substantially similar to a dynamically linked GPL-licensed library. Under Mitel such an API and its accompanying data structures and protocols will be filtered because they are not original. In this situation, dynamically linking to such a GPL-licensed library cannot make the calling program a derivative or infringing work of that GPL-licensed library and hence, cannot engage the viral provisions of the GPL.

Given that there are at least three different scopes of protection for expressive portions of a method of operation, and also given that these approaches can give rise to different outcomes, the potential consequences of dynamically linking to a GPL-licensed library are currently circuit specific.[690] This leaves commercial software vendors in a difficult position since any reasonably successful commercial product is going to be licensed throughout the United States thereby giving a potential plaintiff an ability to select the forum providing the broadest protection to expression embodied within a method of operation – currently the Tenth Circuit.

690. It is possible that different circuits might reach the same conclusion for a particular interface, but for different reasons. For example, under the First Circuit approach, copyright might be denied because the subject matter in question was a method of operation to which the First Circuit will not extend copyright protection. If the same interface was considered in the Tenth Circuit, the Tenth Circuit might be willing to protect expression in that interface, but might not do so for other reasons, such as a lack of originality because the expression was copied from elsewhere or because the expression was so trivial it did not meet the required copyright threshold.

At this time it seems unlikely that the disagreement between the various circuit courts will be resolved soon. The difficulty is that the economics of open source, the potential risk to a commercial product, and the types of resolutions the FSF typically seeks[691] do not favor the occurrence of a test case. The main reason for using open source is to gain access to high-quality software that can provide time-to-market advantages. However, if the use of a particular open source package is going to potentially: (1) compromise a company's proprietary software product through the threat of mandatory licensing under the GPL or loss of copyright protection under §103(a); (2) require litigation all the way to the United States Supreme Court with the associated costs and uncertainty; (3) alienate the open source community; and (4) cause years of consumer fear, uncertainty and doubt about a product, then virtually every company is simply going to make the economically prudent choice and not use GPL-licensed code. Instead, commercial software developers will either write the required code themselves or select another open source package licensed under a non-viral license. As will be discussed below, the FSF's views regarding dynamic linking to GPL-licensed code are probably more reflective of the FSF's preferred outcome than the likely outcome under copyright law. Accordingly, the current lack of cases dealing with the viral provisions of the GPL is probably not due to the strength of the GPL or the FSF's interpretation of copyright law, but instead a matter of economic expediency and prudent decision-making.

5. *Comparison to the FSF Position*

For many years the FSF has stated in its online FAQ that, "[i]f two modules are designed to run linked together in a shared address space, that [sic] almost surely means combining them into one program."[692] More recently, the GPL v.3 has been drafted to include the defined term "corresponding source," which includes "the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communications or control flow between those subprograms and other parts of the work."[693] The

691. Free Software Foundation, *Frequently Asked Questions about the GNU Licenses*, http://www.gnu.org/licenses/gpl-faq.html (last visited Aug. 17, 2010).

692. Frequently Asked Questions about Version 2 of the GNU GPL, *supra* note 320.

693. Free Software Foundation, GNU General Public License, Version 3, June 29, 2007, http://www.gnu.org/licenses/gpl-3.0.txt. This situation is the reverse of that considered in this paper and the FSF seems to be saying that the viral provisions of the GPL extend to any shared libraries called by a GPL-licensed program. As seen in the technical description of dynamic linking, it would be unusual for a dynamically linkable library to be a derivative or infringing work of a calling program since a dynamically linkable library typically does not incorporate any protected expression from a calling program. It is unclear whether the language in the definition of "corresponding source" represents a change in the FSF's views

language in the FAQ is difficult to assess from a copyright perspective since copyright law does not recognize the concept of "combining two works into one." Similarly, the language in the GPL v.3 is similarly difficult to assess because copyright law does not recognize the concept of intimate data communications or flow control. The touchstone for copyright law is whether one work is substantially similar to another work. Therefore, unless the GPL is intended to be a contract – which the FSF explicitly maintains it is not then "combining two works into one" and "by intimate data communications or control flow" must have some meaning other than the simple dictionary definition of those words.

The most plausible meaning for the statements made by the FSF is that they believe if two programs are designed to be dynamically linked at run-time, the calling program is almost certainly a derivative work of the library to which it is linking. However, a detailed examination of the mechanics of dynamic linking and a review of the case law suggests that this statement is overly broad. In the First and Sixth Circuits, the statement is likely to be incorrect for programs that are simply calling dynamically linked routines.[694] If a calling program uses data structures defined in a GPL-licensed library, and if those data structures are used for activities beyond those reasonably related to the invocation of a dynamically linked routine, then the calling program may become a derivative or infringing work of that library. This, however, will only happen if the data structures in question are protectable under copyright, an issue that must be analyzed on a case-by-case basis. In particular, those data structures must meet the copyright standard for originality and must not being filtered by a limiting doctrine such as merger or scènes à faire.

In the Tenth Circuit, the FSF position on dynamic linking fares better because the courts in this circuit are willing to protect expression that may be part of a method of operation at a higher level of abstraction. However, the FSF view that a calling program will "almost surely" be a derivative work of a dynamically linked library remains overly broad because there will be many methods of operation that do not contain any protectable expression. This is particularly so for GPL-licensed libraries implementing third-party defined APIs with third-party defined data structures. In these cases the API and the data structures cannot be used to support a finding of substantial similarity and hence, cannot engage the viral provisions of the GPL. As discussed earlier, when a calling program dynamically links to an object library, none of the library's object code is copied into the calling program. Instead, only the symbol

---

on the scope of the viral provisions of the GPL or is instead an unintended consequence of the newly added term.

694. *See* Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V)*, 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam); Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522 (6th Cir. 2004).

names used by the calling program are copied into its object file. In the case of a GPL-licensed library that implements a third-party developed standard, these symbol names will not be original to the GPL-licensed library. When a calling program also uses data structures from a dynamically linked library, it is also possible that these data structures may be embodied in the object code of that calling program on a non-literal basis. However, once again, these elements will not be original to a GPL-licensed library implementing a third-party developed standard, and a calling program in such a circumstance cannot be a derivative or infringing work of such a GPL-licensed library.

Overall, the position articulated by the FSF seems to be more reflective of their desired outcome than the results suggested by copyright law. Current case law suggests that a calling program may become a derivative or infringing work of a GPL-licensed library to which it links, however, in many cases a calling program will not become a derivative or infringing work of such a library. Each instance of dynamic linking will need to be examined in detail to make this determination. In some cases it will be possible to make a relatively definitive determination, while in many other cases the outcome will be unclear.

E. GPL "Viral" Effects and Inter-process Communication

The final type of program-to-program interaction that will be examined is inter-process communication. This section will examine whether a client program exchanging messages with a GPL-licensed server using a client-server protocol becomes a derivative or infringing work of that GPL-licensed server thereby engaging the viral provisions of the GPL.

In a client-server system, the server will define or select the manner by which clients can request and receive services. This means the server will either define or select the protocols, message types and data structures to be used when communicating with that server. The selected protocols, messages types and data structures are usually defined in a definitions or header file. Alternatively, if a server is implementing a well-known service, such as HTTP or FTP, that server may select appropriate protocol and data structure definitions from a third-party definitions or header file. In each case, the client software will be compiled against the appropriate definitions or header file so the required protocols, messages, and data structure definitions are available for use by the client.

The copyright analysis of inter-process communication in a client-server system is similar to the analysis for dynamic linking. The main difference is that client and a server programs are not linked together at run-time by the operating system. Instead, these programs must use

some other mechanism to establish run-time communication.[695] In a system that relies purely on inter-process communication, the only elements of a server that are embodied in a client will be non-literal. These elements may include data structures, message formats, and protocol sequences. As with dynamically linked programs, no object code from a GPL-licensed server needs to be included in a client program. Unlike dynamic linking, in a pure messaging system the object code version of a client program should not contain any symbols from a GPL-licensed server with which it is communicating.[696] Accordingly, inter-process communication is somewhat easier to analyze than dynamic linking. For inter-process communication, the key question is whether the various protocols, data structures and message formats used to communicate with a GPL-licensed server, and the manner in which those elements are embodied in a client program, make that client program a derivative or infringing work of the GPL-licensed server.

1. *A Derivative Works Analysis For Inter-process Communication*

A determination of whether a client program that communicates with a GPL-licensed server is a derivative or infringing work of that server is always going to be fact dependent. The first question to consider is whether the copied material is recast, transformed, or adapted. It seems relatively easy to conclude that the various protocols, data structures, and message formats are transformed. When these elements are in a definitions or header file, they are in a form that allows those protocols, data structures, and message formats to be included in other programs. Once embedded in those other programs, these protocols, data structures, and messages formats, along with other information and instructions, are used to allow a computer process to communicate with another computer process. Accordingly, the copied material appears to have been adapted to allow it to fulfill a different purpose. In one case, the copied material is in a state that allows its incorporation into other

695. For example, by some mechanism for agreeing on the use of a particular port and IP address.

696. It is possible to create a client-server system in which server defined symbols are used in a client program. In such a scenario, the server may also provide its own library routines that can be used to facilitate communication with that server. Any client wanting to communicate with that server could link to these routines and use them when requesting services. Typically, these types of routines are used to format messages, send requests and receive replies from a server thus removing the need for a calling program to understand low-level details about communication with the server. In these cases, such routines will either be statically or dynamically linked to the client program, and the appropriate GPL analysis will be determined according to the type of linkage used. However, in a pure inter-process communication case, there should be no dynamic or static linking to any GPL-licensed code, instead there should only be an exchange of messages.

programs. In the other instance, the copied material is actually used in executing code.

After satisfying the transformative requirement, the final step in the derivative works analysis is to determine whether the copied material represents a substantial copying of the pre-existing work. This determination is dependent on the scope of protection for client-server protocols and their accompanying data structures. At first glance, interprocess communication appears to be a method of operation. Unfortunately, the cases dealing with methods of operation have generally not provided significant guidance about the meaning of that term. The only case to provide any guidance about the meaning of the term was *Lotus*. In *Lotus*, the term "method of operation" was given a very broad meaning.[697] While there do not appear to be any cases dealing specifically with protocols, other cases, such as *Mitel*, have treated similar technology as a method of operation.[698] Applying the *Lotus* criteria, it would seem that client-server protocols can qualify as methods of operation. A client-server protocol is the means by which a client program accesses or requests services provided by a server.[699] Without a client-server protocol, a programmer wanting to take advantage of the services provided by a server would not be able to send requests or receive responses from that server. The entire purpose of a client-server protocol is to allow other processes to interact with a server to take advantage of the capabilities provided by that server. Accordingly, it seems reasonable to conclude that under the *Lotus* criteria, a client-server protocol qualifies as a method of operation.

An examination of the dictionary meaning of the constituent words of the term "method of operation" leads to a similar conclusion. The Merriam-Webster Online Dictionary defines "method" as: "a way, technique, or process of or for doing something;" "a procedure or process for attaining an object;" or "a systematic procedure, technique, or mode of inquiry employed by or proper to a particular discipline or art."[700] The same dictionary defines "operation" as "a method or manner of functioning," "performance of a practical work or of something involving the practical

697. Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807, 815 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996) (per curiam).

698. *See* Secure Services Tech., Inc. v. Time &Space Processing Inc., 722 F.Supp. 1354 (E.D. Va. 1989). This case dealt with certain variations to the T.30 protocol and copyright protection was ultimately denied because of a lack of originality. *Id.* at 1363. In passing the Court observed that "Copyright protection for the timing of electronic binary signals is precluded by the copyright laws' exclusion of 'any idea, procedure, process, system, method of operation, concept, principle or discovery.' Timing is nothing more than the process by which electronic signals are created, transmitted, received." *Id*. at 1363 n.25.

699. VikontSolutions Corp., Glossary: Client-Server Protocol, http://www.vikont.com/clients/glossary.htm (last visited Apr. 26, 2009).

700. Merriam-Webster Online Dictionary, Method, *supra* note 652.

application of principles or processes;" or "any of various mathematical or logical processes (as addition) of deriving one entity from others according to a rule."[701] Other English and legal dictionaries have similar definitions.[702] A client-server protocol is variously described as, "the manner in which clients request information and services from a server and also how the server replies to that request,"[703] and "[a] communication protocol between networked computers where the services of one computer (the server) are requested by the other (the client)."[704]Hence, an examination of the dictionary meaning of the constituent words of the term "method of operation" and the typical industry meanings for the term "client-server protocol" suggests that a client-server protocol is a method of operation.

As discussed in Section II of this paper during the analysis of dynamically linked APIs, there seem to be three different ways of determining the scope of protection for methods of operation; these are the approaches taken in *Lotus*, *Mitel* and *Lexmark*. As previously discussed, it appears that the use of these approaches can lead to different results.

Under *Lotus*, the fact that expression may be embodied in a method of operation is immaterial. The initial inquiry is whether a client-server protocol is a "method of operation." If so, no further inquiry is required because under §102(b) any expression embodied in that method of operation that is required for its use will be denied copyright protection. Accordingly, under *Lotus* any expression embodied in a client-server protocol that is required for its use will not contribute to a finding of substantial similarity and hence, will not make a client program a derivative or infringing work of a server program with which it wishes to communicate.

Under a *Lotus*-based approach, a client-server protocol will almost certainly be found to be a method of operation. As previously discussed, the First Circuit interpreted the term "method of operation" to refer to "the means by which a person operates something, whether it be a car, a food processor, or a computer."[705] The court further observed that without the menu commands and menu command hierarchy, users would not be able to access and control the Lotus 1-2-3 functional capabilities.[706] The purposes served by the Lotus 1-2-3 menu commands and menu command hierarchy are very analogous to the purposes served by a client-server protocol. The messages a server program is willing to receive and

701. Merriam-Webster Online Dictionary, *Operation*, *supra* note 653.

702. *See* King, *supra* note 654.

703. Client/Server Frequently Asked Questions, 2.6 What is Middleware?, http://www.faqs.org/faqs/client-server-faq/ (last visited Apr.26, 2009).

704. VikontSolutions Corp., *supra* note 699.

705. Lotus Dev. Corp. v. Borland Int'l, Inc. (*Lotus V*), 49 F.3d 807, 815 (1st Cir. 1995).

706. *Id.*

process are the means by which a client program accesses the functionality provided by a server. Without a client-server protocol, client programs would not be able to access the capabilities made available by a server. As discussed earlier, it is unlikely that the First Circuit intended to restrict the term "method of operation" to those instances in which a human operator is involved.

Accordingly, the fact that services are being requested by one program from another is unlikely to change a *Lotus*-based analysis. Under *Lotus*, once a particular client-server protocol is classified as a method of operation, the analysis is straightforward – copyright protection is denied under §102(b) for any expression embodied in that client-server protocol that is necessary for its use. This expression cannot be used as a basis for substantial similarity. Hence, under a *Lotus*-based approach communication with a GPL-licensed server should not engage the viral provisions of the GPL.

This result, however, does not conclude the substantial similarity analysis. It is still possible that other subject matter used to communicate with a GPL-licensed server may support a finding of substantial similarity. Inter-process communication is similar to dynamic linking in that no object code is copied from the program with which the interaction is occurring. However, the substantial similarity test can still be satisfied through the copying of non-literal elements. Once again, the most likely type of non-literal expression that may be copied will be the data structures used to describe the messages exchanged between a client and server.

The use of data structures within a client-server protocol to exchange complex data is very common.[707] In order to satisfy the substantial similarity test, such data structures will have to be sufficiently original and not denied copyright protection under any limiting doctrines. Under *Lotus*, the first limiting doctrine to consider is §102(b).[708] If a particular data structure is part of a client-server protocol and hence part of a method of operation, then under *Lotus* any expression embodied in that data structure that is necessary for use of the method of operation should not be protected by copyright. Therefore, under *Lotus* the use of potentially expressive data structures within client-server messages should not result in copyright infringement.

However, if a data structure is used within a client-server message it is very common for that data structure to be used in other parts of a client program. If such a data structure is used for purposes unrelated to

707. For example, in the case of the UNIX *sockets* protocol, the definition of a *socket* is an integral part of that protocol and if use of the *socket* data structure was prohibited for copyright reasons, then the ability to use the whole protocol would be compromised.

708. *Lotus V.* 49 F.3d at 815.

communication with a server process or exploitation of the responses provided by a server then those unrelated uses can potentially support a finding of substantial similarity. The reason for this distinction is that a prohibition on these uses will not prevent the use of the client-server protocol and messaging system and thereby create a *de facto* monopoly over the protocol and messaging system. However, a court should permit a sufficiently broad range of uses of the data structures embedded in a client-server protocol and/or messaging system to allow a client program to receive the full benefit of the services provided by a server program. Only uses that are not reasonably related to the receipt or use of server responses should support copyright infringement.

The next issue to consider in determining whether the use of data structures embedded in a client-server protocol can support a finding of substantial similarity is originality. The originality requirement has two aspects. The first is the requirement to meet the copyright standard for originality as articulated in *Feist*.[709] The originality standard is very low and requires only that there is some "creative spark."[710] Many data structures used in a GPL-licensed client-server protocol or messaging system will meet this low standard. However, there undoubtedly will be some minimal data structures that do not meet this standard.[711]

The more interesting aspect of originality is the requirement that the subject matter must not been copied. There will be many data structures used in client-server protocols or messaging systems that have been copied from elsewhere and these data structures will not be original when used in subsequently-developed servers or protocols.[712] A copied data structure cannot be used to establish substantial similarity. This result is very important because there are literally hundreds if not thousands of packages built on the data structures defined by the UNIX operating system and the various communications protocols used with that operating system.

Thus, under *Lotus* there seems to be a reasonably broad ability to exchange messages with GPL-licensed servers without engaging the viral provisions of the GPL. However, because *Lotus* is only binding in the First Circuit, the approaches taken in other circuits must also be considered.

The next approach to consider is that taken by the Sixth Circuit in

709. Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340 (1991).

710. *Id.* at 345.

711. *Id.* at 350 ("There remains a narrow category of works in which the creative spark is utterly lacking or so trivial as to be virtually non-existent").

712. For example, in the case of a server providing email capabilities, any data structures dealing with message formats or mailbox formats may simply be copied from existing non-GPL licensed specifications or implementations.

Lexmark.[713] When applied to client-server protocols and associated messaging systems, the tests enunciated by the Sixth Circuit lead to results similar to those under *Lotus*. Under the *Lexmark* minority approach when potentially copyrightable expression is used as a necessary part of a method of operation that use will not be infringing because of the defense of merger with a method of operation. In such a situation, since there is no infringement, a client program cannot become a derivative work of a GPL-licensed server by exchanging messages with that server.

However, the minority opinion in *Lexmark* makes it clear that copyrightable expression that is not protected in the context of a method of operation may still be protected in other contexts.[714] Accordingly, if expressive parts of a client-server protocol and messaging system are used in other parts of a client program in contexts unrelated to the receipt of services from a GPL-licensed server, those uses can support a finding of substantial similarity, thus making a client program a derivative or infringing work. Substantial similarity can arise from these non-related uses because the minority in *Lexmark* held that merger with a method of operation is a defense to infringement and does not act as a bar to copyrightability.[715] As discussed above in the *Lotus* analysis, expression from a client-server protocol and its related messaging system will generally need to be used in other parts of a client program to allow the client program to receive the full benefit of the services provided by a server. These types of uses should not be viewed as uses in another context, and the defense of merger with a method of operation should still be available so that full exploitation of the capabilities provided by a server is not affected.

The scope of protection for a client-server protocol and its related messaging system under the majority approach in *Lexmark* is similar to that under the minority approach. However, because the majority does not appear to have taken a firm position on whether merger is a bar to copyright or simply a defense to a claim of copyright infringement, the exact bounds of protection are not as clear as those under the minority approach. As with the minority approach, any expression included in a method of operation and required for the use of that method of operation will not be infringing.[716] Accordingly, a client program should not become a derivative or infringing work of a GPL-licensed server with which it is communicating simply because of the exchange of messages.

---

713. Lexmark Int'l, Inc. v. Static Control Components, Inc., 387 F.3d 522 (6th Cir. 2004).

714. *See Id.* at 557-58.

715. *Id.* at 557.

716. *Id.* at 537-38.

Furthermore, this type of communication should not cause the viral provisions of the GPL to be engaged. However, since the majority opinion does not make it clear whether merger is a defense or a bar to copyrightability, it is not clear whether a client program's use of the expressive aspects of a client-server protocol and its messaging system in other contexts can support a finding of substantial similarity. In this circumstance it is unclear whether a client program can become a derivative or infringing work of a GPL-licensed server with which it is communicating.

If merger with a method of operation is a bar to copyright, then the expressive portions of a client-server protocol and its related messaging system can be used in other contexts and such uses will not cause a client program to become a derivative work. If merger is a defense to an allegation of copyright infringement, then use of protectable expression in other contexts within a client program may cause that program to become a derivative or infringing work and thus engage the viral provisions of the GPL. If this is the case, then the scope of protection accorded by the majority in *Lexmark* is the same as that envisioned by the minority.

In summary, under *Lexmark* the simple exchange of messages with a GPL-licensed server should not cause the viral provisions of the GPL to become applicable to a client program. Under the minority approach, other uses of expressive parts of a client-server protocol and messaging system may engage the viral provisions of the GPL. In particular, data structures defined in a GPL-licensed server that are an integral part of the client-server protocol for that server pose a particular risk because such data structures are commonly used for functions beyond simple messaging. It seems reasonable to conclude that uses of expressive material for the purpose of receiving the full benefit of the services provided by a GPL-licensed server should be covered by the merger defense and should not cause a client program to become a derivative work of a GPL-licensed server with which it is communicating. However, this issue was not considered in the minority opinion and accordingly this conclusion is speculative. It is, however, clear that the use of expressive parts of a client-server protocol and its related messaging system in contexts unrelated to the utilization of services provided by a GPL-licensed server can support a finding of substantial similarity and can cause a client program to become a derivative or infringing work of the GPL-licensed server from which such expression is being copied. The question of whether the use of expressive subject matter from a client-server protocol and related messaging system will support a finding of substantial similarity will always be fact dependent.

In general, the results under the majority approach in *Lexmark* should be similar to those under the minority approach. The main differ-

ence is the treatment of expressive material embodied in a method of operation when that material is used outside the context of the method of operation. Under the majority approach the treatment is more uncertain. If the majority intended merger to act as a bar to copyrightability, then the use of expressive material embodied in a client-server protocol and related messaging system is unfettered and cannot support a finding of substantial similarity. If the majority intended merger to act only as a defense to copyright infringement then the scope of protection for expression embodied in a method of operation will be the same as that under the minority approach. Neither the majority approach nor the minority approach considered use of expression for the purpose of gaining the full benefit of the services provided by a GPL-licensed server and accordingly it is unclear under both approaches whether such uses will support a finding of substantial similarity and engage the viral provisions of the GPL.

The last circuit to consider is the Tenth Circuit's analytical approach as set forth in *Mitel v. Iqtel*. As discussed earlier, while the *Mitel* approach is conceptually easy to understand, in practice, it is perhaps the most difficult to apply. Under *Mitel*, any expression within a method of operation that is necessary for the use of that method of operation can give rise to an enforceable copyright.[717] This differs from *Lotus* and *Lexmark* since neither case provides protection for expression required for the use of a method of operation. Accordingly, it appears that under *Mitel* it is possible for a simple exchange of messages to support a finding of substantial similarity. Therefore, if a client program utilizes copyrightable expression from a client-server protocol or its related messaging system, this use can potentially make the client program a derivative or infringing work of a GPL-licensed server with which it is communicating, and thereby engage the viral provisions of the GPL.

As was the case with dynamic linking, the inherent uncertainty of the Abstraction-Filtration-Comparison test will make it difficult for commercial entities to draw definitive conclusions about the use of client-server protocols and their related messaging systems. It appears that there are likely to be significantly fewer unencumbered protocols and messaging systems under the *Mitel* approach than under the approaches taken by the First and Sixth Circuits. In the First and Sixth Circuits, the mere exchange of messages with a GPL-licensed server cannot make a client program a derivative work of a GPL-licensed server. This is not the case in the Tenth Circuit where a simple exchange of messages can potentially make a client program a derivative or infringing work of a GPL-licensed server with which it is communicating. It is also more likely under *Mitel* that the use of data structures embodied within a cli-

717. *See* Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366 (10th Cir. 1997).

ent-server protocol or related messaging system will support a finding of substantial similarity.

If *Mitel* is ultimately accepted as the correct approach for determining the scope of protection for client-server protocols and messaging systems, there will still be situations where a client program will not become a derivative or infringing work of a GPL-licensed server with which it is communicating. However, given the potential severity of the GPL-related consequences for a commercial software product and given the uncertainty in applying the *Mitel* test, if *Mitel* becomes the accepted approach for determining the scope of protection for methods of operation then it seems unlikely that many companies will risk using GPL-licensed servers within their commercial offerings.

As with dynamic linking, however, there appears to be one situation where this may not be the case. When a GPL-licensed server uses a third-party defined protocol based on third-party defined data structures, the potential risk posed by the viral provisions of the GPL seems to be significantly lower. In this special case, the client-server protocol and data structures will not be original to the GPL-licensed server because the author of the GPL-licensed server will have copied them. In this case, even under the *Mitel* approach, an exchange of messages with this type of GPL-licensed server and the use of the related data structures cannot make a client program substantially similar to that GPL-licensed server, and hence cannot make the client program a derivative or infringing work of that GPL-licensed server or engage the viral provisions of the GPL.

Given that there are at least three different ways of determining the scope of copyright protection for methods of operation, and also given that these different approaches can lead to significantly different results, a determination about whether a client program is a derivative or infringing work of a server with which it is communicating is circuit specific. As with dynamic linking, this places commercial software vendors in a difficult position since any reasonably successful commercial product is going to be licensed throughout the United States. Accordingly, a potential plaintiff will be able to bring an action in the forum that provides the broadest protection to client-server protocols and messaging systems – currently the Tenth Circuit.

An examination of the case law in the various circuits, including even the Tenth Circuit, suggests that there are certain types of inter-process communication that should not engage the viral provisions of the GPL. However, as with dynamic linking, the lack of uniformity between the various circuit courts, the ambiguity of certain statements made by the FSF about inter-process communication with GPL-licensed software, the FSF's reputation for enforcement, and the significant consequences arising from an application of the GPL to a commercial software pro-

gram, probably means that commercial software developers are going to be reluctant to test the boundaries of inter-process communications between proprietary software and GPL-licensed software.

2. *Comparison to the FSF Position on Inter-process Communication*

In its online FAQ, the FSF says the following about inter-process communication:

> By contrast, pipes, sockets and command line arguments are communication mechanisms normally used between two separate programs. So when they are used for communication, the modules are normally separate programs. But if the semantics of the communication are intimate enough, exchanging complex internal data structures, that too could be a basis to consider the two parts as combined into a larger program.[718]

First, it should be noted that the FSF acknowledges that inter-process communication is less likely to engage the viral provisions of the GPL than static or dynamic linking. Indeed, the first sentence seems to indicate that it would be unusual for inter-process communication to engage the viral provisions of the GPL. Unfortunately, the very next sentence injects a high degree of uncertainty about the type of inter-process communication that the FSF considers to be viral. In particular, trying to determine what the FSF considers to be "too intimate" or "too complex" is virtually impossible.

As discussed in the dynamic linking section, the terminology used by the FSF is very difficult to analyze from a copyright perspective since copyright law does not contemplate matters such as the intimacy of communication or combining two parts into a larger program. If the FSF's statements about inter-process communication are converted into copyright law concepts, it appears they are likely to be over inclusive in some situations and under inclusive in others. For example, the FSF statements are probably over inclusive in situations where a client program communicates with a GPL-licensed server using a highly complex protocol based on highly complex data structures that are not original to the GPL-licensed server.[719] Hypothetically, under the FSF criteria, the exchange of complex internal data structures and the high degree of interaction between the two programs would engage the viral provisions of the GPL. However, under a copyright analysis, since the data structures and protocol are not original to the GPL-licensed server, they cannot support a finding of substantial similarity and thus cannot make a client program a derivative or infringing work of such a server. Accordingly,

718. Frequently Asked Questions about Version 2 of the GNU GPL, *supra* note 320.

719. Such a situation can arise if a GPL-licensed server is implementing capabilities defined in an RFC.

unless the FSF is relying on contractual principles,[720] the FSF's characterization of the breadth of the viral provisions of the GPL for this type of situation is overly broad. If the use of the term "internal data structures" is only meant to refer to original data structures defined within a GPL-licensed program, then the FSF's position is much more aligned with the expected results under copyright law.

The statements made by the FSF can also be under inclusive in certain situations. In particular, a client may communicate with a server using a protocol and data structures that are not particularly complex, but still be a derivative or infringing work of that server. Such a protocol and data structures may still be sufficiently original to meet the very low standard required by copyright law.

In the Tenth Circuit, despite the fact that expression may be embodied in a method of operation at a higher level of abstraction, that expression may still be protected and can potentially support a finding of substantial similarity thereby rendering a client program a derivative or infringing work of such a server. In this scenario, the FSF suggests that the viral provisions of the GPL will not be engaged. However, a more copyright-oriented approach suggests that in at least the Tenth Circuit such a client program may be a derivative or infringing work that engages the viral provisions of the GPL.

These examples demonstrate how the imprecise language used by the FSF makes it difficult to predict FSF's potential reaction to any particular instance of inter-process communication with GPL-licensed code. The FSF could do the industry a great service by clarifying its statements about the viral provisions of the GPL. In particular, it would be very useful if the FSF articulated its views in terms recognized by copyright law with supporting case law references.

Absent any clarification from the FSF, a reasonable approximation of the FSF's position seems to be that a client program will engage the viral provisions of the GPL if it uses non-trivial data structures and protocols in its communication with a GPL-licensed server. If this is the case, then, at least in the First and Sixth Circuits, this position is likely to be incorrect in situations where a client program is simply exchanging messages with a GPL-licensed server. If a client program uses data structures defined in a GPL-licensed server, and if those data structures are used for functions beyond those reasonably related to the receipt of services from that server, then the client program may be a derivative or infringing work of that server. However, this determination will be fact dependent and subject to the copyright originality requirement and traditional limiting doctrines.

720. The FSF has consistently maintained it is not relying on contractual principles.

In the Tenth Circuit, the FSF's position on inter-process communication matches the results suggested by recent jurisprudence more closely. This is because of the Tenth Circuit's apparent willingness to protect expression that may be a part of a method of operation at a higher level of abstraction. However, in every circuit there is a significant exception for GPL-licensed servers that implement third-party defined protocols with third-party defined data structures. In this special case, the FSF's position is once again overly broad. Since the viral provisions of the GPL cannot apply to a client program no matter how intimate the communication and no matter how complex the data structures.

As with dynamic linking, each instance of inter-process communication will have to be examined in detail to determine whether communication with a GPL-licensed server will engage the viral provisions of the GPL. As a general proposition, the statements made by the FSF with respect to the viral effects of inter-process communication are more accurate than the statements made by the FSF about dynamic linking. However, certain statements still appear to be overly broad.

IV. CONCLUSION

There are significant philosophical and economic differences between the free software community and the commercial software industry. However, the shear volume of free software being developed, its increasing use by significant user communities, and the increasing cost and time-to-market pressures experienced by commercial software developers means that the temptation to use free software will be increasingly difficult to resist.

Additionally, at a technical level, there are going to be more ways for proprietary software and free software to interact. To date these technical interactions have generally involved: static linking, dynamic linking, or inter-process communication. From a product perspective, the ability to interoperate with free software can be very beneficial for a commercial software vendor because its use can allow that developer to bring more products to market, more quickly, at a lower cost, and with improved quality. However, when commercial proprietary software interacts with free software licensed under the GPL, there is a distinct possibility that such use will be highly detrimental from a commercial perspective. Indeed, certain interactions with GPL-licensed software can potentially cause a commercial software vendor to lose the ability to control the exploitation of its product, and, as a result, waste millions of dollars spent on research, development, and marketing, and even worse, forgo significant future revenues. Even in a best-case scenario where the FSF is willing to allow a developer to remove GPL-licensed code from its product, the resulting development and replacement costs, plus the down-

stream costs of deploying the re-engineered product to existing customers, can result in significant costs, delays and customer disruption.

Because of the well-publicized risks associated with the use of or interaction with GPL-licensed software, there has been much speculation about the circumstances under which the viral provisions of the GPL may be applicable. To date, there has been no consensus on this issue. This paper has examined the three most common ways in which programs may interact from a detailed technical perspective. This paper has also examined the recent jurisprudence in those areas of copyright law that appear to be most relevant to these types of program-to-program interactions. The principles set forth in this jurisprudence have been applied to static linking, dynamic linking, and inter-process communications with the goal of identifying those instances where the viral provisions of the GPL are likely to be engaged and those instances where they are unlikely to be engaged. In each case, these results were compared to the positions articulated by the FSF.

The results of this exercise are mixed. An examination of the relevant case law suggests there are two important areas of copyright law to consider when determining the potential scope of the viral provisions of the GPL. The first area relates to the scope of the derivative work right as applied to computer software. The second area relates to the protection to be provided to expression embodied within a method of operation. These areas of copyright law contain significant uncertainties.

In the case of the derivative works right, the uncertainty arises from a general lack of case law dealing with the scope of this right and a specific lack of case law dealing with the scope of this right as applied to computer software. In the case of methods of operation, the uncertainty is caused by a significant disagreement between various circuit courts about the proper scope of protection. The imprecise drafting of the GPL and various statements made by the FSF and others in the free software community further compound this uncertainty.

Unfortunately, at this time it seems unlikely that these issues are going to be resolved soon. The economics of free software and the potential risk to a commercial product seem unlikely to generate a good test case. A commercial software developer uses free software or open source software to gain access to good quality software that can help provide time-to-market advantages. In many cases, a commercial software developer will have a variety of free or open source packages to choose from. In most cases, a commercial software developer will also have the option of developing the necessary functionality in-house or licensing a similar commercial product. Accordingly, most commercial software vendors will simply make the economically prudent choice not to use GPL-licensed software if the use of a free software package is going to

potentially: (1) compromise a company's proprietary software product through mandatory licensing under the GPL; (2) cause a loss of copyright protection under §103(a); (3) require a company to litigate all the way to the United States Supreme Court; (4) alienate the free software and open source communities; and (5) cause years of consumer fear, uncertainty, and doubt about the company's now impugned product(s). All of these factors suggest that no commercial software vendor will want to be the test case for any GPL-related issues. Additionally, it appears unlikely that the FSF is going to clarify its position on the scope of the viral provisions of the GPL. Accordingly, the economics and practicalities of free software use within commercial software products seem destined to dictate a *status quo* of uncertainty.

Notwithstanding the uncertainty about the applicability of the viral provisions of the GPL, there are a few conclusions that can be drawn with some degree of confidence. First, static linking to a GPL-licensed library will almost always engage the viral provisions of the GPL. Second, dynamically linking to a GPL-licensed library does not necessarily engage the viral provisions of the GPL under any current jurisprudence. Certain circuits, most notably the First and Sixth Circuits, have rendered decisions that suggest that dynamic linking should not engage the viral provisions of the GPL in a wide range of circumstances. Other circuits, most notably the Tenth Circuit, have rendered decisions that can, but depending on the particular API under consideration, do not necessarily cause the viral provisions of the GPL to become applicable to a dynamically linking program. Further, in all circuits there seems to be a commercially significant class of GPL-licensed software, namely software that implements third-party defined APIs using third-party defined data structures, which seems unlikely to engage the viral provisions of the GPL.

Finally, the use of inter-process communication between a client and a GPL-licensed server does not necessarily engage the viral provisions of the GPL under any current jurisprudence. Once again, certain circuits, most notably the First and Sixth Circuits, have rendered decisions that suggest that exchanging messages with a GPL-licensed server should never engage the viral provisions of the GPL. By contrast, in the Tenth Circuit, exchanging messages with a GPL-licensed server can, but depending on the particular client-server protocol, messaging system, and data structures, does not necessarily cause the viral provisions of the GPL to become applicable to a client program communicating with that GPL-licensed server. As with dynamic linking, there seems to be a commercially significant class of GPL-licensed servers, namely those servers that implement third-party defined client-server protocols and messaging systems using third-party defined data structures, which are un-

likely to engage the viral provisions of the GPL for client programs communicating with those servers.

Beyond these rough heuristics, the one thing that can be said with complete confidence about the viral effects of the GPL is that any commercial software developer wanting to test the outer boundaries of the GPL will be taking a significant risk that may lead to significant adverse consequences. Further, the more a program interacts with free software, the greater the chance that such use will be challenged; the greater the cost of defending or undoing such use will be; and the greater the potential intellectual property, licensing, and commercial consequences will be.