IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF COLUMBIA

STATE OF NEW YORK, *et al.*,	)	
Plaintiffs,	)	
v.	)	Civil Action No. 98-1233 (CKK)
MICROSOFT CORPORATION,	)	
	)	Next Court Deadline: April 2, 2002 Remedies Hearing
Defendant.	)	

DIRECT TESTIMONY OF DR. ANDREW W. APPEL

(REMEDIES 1, 4, 16)

Table of Contents

I. INTRODUCTION

1. My name is Andrew W. Appel. I am a professor of computer science at Princeton University.

2. I have been asked to review the proposed remedies set forth by Microsoft and the Litigating States. Based on my review and analysis, I will evaluate the technical implications of each remedy proposal[1].

3. In addition to reviewing both remedy proposals, I have consulted various documents and individuals to gain further insight into particular issues relevant to the remedy proposals. I have also supervised an effort to review parts of Microsoft's source code in order to gather information relevant to my expert opinion. The following is a summary of my opinions, explained in further detail in the paragraphs below:

4. I will first provide a general overview of the technologies relevant to my testimony, including the typical software components of a personal computer ("PC") and the role and significance of middleware as a means of allowing software applications to "port," *i.e.*, to operate on various PC operating systems.

5. I will then discuss the necessity of disclosing sufficient information – including application programmer interfaces ("APIs") and technical data – to allow software

[1] For purposes of my testimony, I will use the term "Microsoft's Remedy" to refer to Microsoft Corporation's Second Revised Proposed Final Judgment and the term "States' Remedy" to refer to the Plaintiffs' First Amended Proposed Final Judgement.

programs to interoperate with platform software such as operating systems and middleware.

6. I then apply these principles by comparing and analyzing the disclosure provisions of Microsoft's and the States' Proposed Remedies. In comparison to the States' disclosure provisions, I find Microsoft's disclosure provisions to be lacking in a number of important respects:

- Microsoft's Remedy does not require the disclosure of sufficient technical data to instruct a software developer how to make effective use of the APIs disclosed by Microsoft platform software;
- Microsoft's Remedy does not require the disclosure of important APIs exposed by Microsoft middleware, such as Internet Explorer and Office;
- Microsoft's Remedy does not require the disclosure of APIs that many of Microsoft's middleware products use to interoperate with Windows – and therefore competing middleware products are unable to make use of those APIs needed to run on Windows;
- Microsoft's Remedy precludes anyone from using the limited disclosures required of Microsoft in developing competing software products; and
- Microsoft's Remedy does not allow for an often necessary resource in creating software that runs on Microsoft software, namely, the ability to review the relevant portions of the Microsoft code.

7. In my depositions, I was questioned as to whether the disclosures required by the States' Remedy would permit the copying of Windows. The answer is that it will not – the

States' Remedy will require only the disclosure of a small amount of data needed to allow various software programs to interoperate effectively. My testimony addresses this issue.

8. In addition to the limited nature of the disclosures mandated by Microsoft's Remedy, there is another significant problem with these provisions. Section III.J.1 of Microsoft's Remedy creates a broad exemption from any disclosure, purportedly on grounds of computer security. As an expert in computer security issues, I will testify that this exemption is technically unjustified and threatens virtually to eliminate even the limited disclosures provided under Microsoft's Remedy.

9. My testimony also supports the technical feasibility of section 1 of the States' Remedy, which would require Microsoft to create an "unbound" version of Windows from which previously commingled middleware can be removed. It is my opinion that this remedy provision is technically feasible, based in part on the fact that Microsoft has already performed substantial parts of this engineering task in creating its Windows XP Embedded operating system product.

10. Next, I address a strange and unnecessary provision of Microsoft's Remedy (section III.H.2.b) that would appear to give Microsoft the unilateral discretion to override a user's choice of non-Microsoft middleware (such as the Netscape browser) and replace it with Microsoft middleware (such as Internet Explorer).

11. Finally, I offer testimony in support of the States' Remedy proposal 16, which would require Microsoft to support established industry standards for interoperability in circumstances when the company claims to have done so.

II. BACKGROUND AND QUALIFICATIONS

12. I have been a professor of computer science at Princeton University for over fifteen years and I currently hold the position of Associate Chair of the Princeton University Department of Computer Science.[2]

13. My experience in the field of computer science dates back to 1972 when, at the age of 12, I started programming computers. During the following years, I spent a great deal of time at the University of Illinois Urbana-Champaign computer labs where I was able to learn about, and practice, programming computers.

14. By age 16, I obtained my first job in the field of computers, when the University of Illinois Medical School hired me to assist them with the development of medical informatics—the application of computer technology to manage information for medical care, education, and research. I spent five summers working on informatics for the medical school.

15. I attended Princeton University for my undergraduate studies. Having already gained a great deal of experience with computers prior to college, I decided to major in physics. For my senior thesis, I combined my expertise in computer science and physics to develop a new computer algorithm for performing astrophysical calculations. I received a prize in recognition of my thesis, which was published in a journal and is still influential today. In 1981, I received my A.B. in physics, *summa cum laude*, from Princeton.

[2] My full *curriculum vitae* is attached as Exhibit A.

16. After graduating from Princeton, I entered the doctoral program in computer science at Carnegie Mellon University on a National Science Foundation fellowship. My graduate research ranged from continued work on astrophysical algorithms to the development of programs that could automatically play games, such as Scrabble. I also spent two summers working for a firm in New York developing financial forecasting software.

17. For my dissertation, I researched and developed a program that automatically derived compilers from suitably specified programming languages. One of the key prerequisites for the success of this work was the sufficient specification and documentation of programming languages. This shows that since early in my career I have been interested in the specification of software modules and interfaces, a concept of disclosure and documentation related to those presented in this case.

18. In 1985, I received my Ph.D. in computer science from Carnegie Mellon. I spent the better part of the following year working on a joint project between IBM and Carnegie Mellon helping to develop the Andrew[3] system, an early research effort in the development of networked file systems and desktop window systems. My work focused in particular on the development of the window manager and the creation of demonstration applications for the window system. This gave me early experience with network file systems and with windowing "desktop" user interfaces for operating systems, technologies related to this case.

19. In 1986, I joined the Princeton University computer science department. My research interests since joining the department have included programming languages, computer

[3] Named after Andrew Carnegie and Andrew Mellon.

security, compilers, type theory, semantics, software engineering, and automated theorem proving. I have written textbooks and more than sixty journal and conference papers on these and related computer science topics, and I have taught undergraduate and graduate courses in these topics now for over fifteen years.

20. I have focused much of my academic research and writing on the technologies that are now used in middleware, in particular component software platforms such as the Java Runtime Environment and Microsoft's Common Language Runtime ("CLR"). I started working jointly with Bell Labs in 1986 on the development of Standard ML of New Jersey—a compiler and run-time system employing many of the same concepts as current middleware technologies such as Java and CLR. My work on such component software platforms continues today through research projects involving Java and CLR.

21. I have many years of experience as a software developer for both commercial and academic research purposes. In recent years, I have primarily focused on the software development of compilers (programs that translate source code into executable code), automated memory management (particularly the form of memory management called "garbage collection"), and run-time systems, all of which are important to middleware technologies such as Java and CLR.

22. I have received recognition for my work by the Association for Computing Machinery (ACM), the largest and most prestigious professional society in the field of computer

science. In 1993, I was named Editor in Chief of the research journal *ACM Transactions on Programming Languages and Systems,* and in 1998 I was named an ACM Fellow.[4]

23. Although I offered testimony in a previous case that could be characterized as a combination of fact and opinion testimony, this is my first experience being retained as an expert. I chose to serve as an expert in this case because of my firm belief in the importance of the issues presented in this hearing and because I have considerable relevant expertise to assist in explaining a number of the case's technical issues.

III. BASES FOR MY OPINIONS

24. In addition to my expertise and training, I have considered multiple materials in forming my opinions in this case:

   a. Plaintiffs' First Amended Proposed Final Judgment ("States' Remedy").

   b. The Second Revised Proposed Final Judgment ("Microsoft's Remedy") between Microsoft Corporation, the United States of America, and the States of New York, Ohio, Illinois, Kentucky, Louisiana, Maryland, Michigan, North Carolina and Wisconsin.

   c. The ruling of the United States District Court for the District of Columbia in *United States v. Microsoft Corp.*, including the Findings of Fact, 84 F. Supp. 2d 9, Conclusions of Law, 87 F. Supp. 2d 30, and June 7, 2000 Memorandum and Order, 97 F. Supp. 2d 59.

[4] The ACM Fellows program "recognize[s] and honor[s] outstanding ACM members for their achievement in computer science and information technology and for their significant contributions to the mission of the ACM."

d. The June 28, 2001 decision of the United States Court of Appeals for the District of Columbia Circuit in *United States v. Microsoft Corp.*, 253 F.3d 34.

e. The written direct testimony of Edward W. Felten in the liability proceedings of *United States v. Microsoft Corp.*

f. I consulted portions of certain reference materials and textbooks for background information, including Peter G. Aitken, *Developing Solutions With Office 2000 Components and VBA* (Prentice Hall PTR 2000); Paul McFedries, *VBA for Microsoft Office 2000 Unleashed* (Sams Publishing, 1999); and Stephen Forte, Tom Howe, and James Ralston, *Microsoft Access 2000 Development Unleashed* (Sams Publishing, 1999).

g. I reviewed the Supplemental Response of the Plaintiff Litigating States' to Defendant Microsoft Corporation's First Set of Written Interrogatories and Responses of Plaintiff Litigating States to Defendant Microsoft Corporation's Second Set of Written Interrogatories.

h. Microsoft Corporation's Response to Plaintiff Litigating States' First Set of Interrogatories in Remedy Proceedings, served January 11, 2002.

i. I had conversations with a number of individuals within the computer science community including other academics and those in the computer industry. For example, I have had conversations with Jeremy White (president of CodeWeavers.com), Miguel de Icaza (a software developer at Ximian.com), and Jay Lepreau (a Research Associate Professor of Computer Science at the University of Utah).

j. I read “Microsoft’s Digital Rights Management Scheme – Technical Details” by Beale Screamer (pseud.), published October 18, 2001 at cryptome.org.

k. I read U.S. Patent No. 6,330,670, “Digital rights management operating system,” assigned to Microsoft Corporation, issued on December 11, 2001.

l. I read a number of relevant articles, including “Open Source Comes to .Net,” by Justin Hibbard, *Red Herring,* October 1, 2001; and “Will Microsoft Mono-polize open source?,” by Jim Landgrave, *Tech Republic,* August 23, 2001.

m. I consulted “The Flux OSKit: A substrate for kernel and language research,” by Bryan Ford, Godmar Back, Greg Benson, Jay Lepreau, Albert Lin, and Olin Shivers, which appeared in *Proceedings of the 16th ACM Symposium on Operating Systems Principles,* October 1997.

n. I reviewed literature on, and am examining, the Microsoft product Windows XP Embedded.

o. I am directing an ongoing effort to review parts of Microsoft’s source code for Windows XP Professional.

p. I have read the deposition testimony and am reading the ongoing trial testimony of a number of witnesses in this case, and I make specific references to portions of such testimony herein. I have also referenced certain other bases for my opinions in the context of the particular testimony that I offer below.

IV. OVERVIEW OF TECHNOLOGY

25. I believe that it is helpful to begin my testimony by describing and explaining some of the relevant computer science technologies and terminology that I will be discussing.

26. An operating system is software that manages and controls a computer's hardware and provides a platform on which application programs (or middleware) can run. The operating system provides abstraction and protection of hardware resources. Abstraction is the hiding of low-level details; it is necessary (for example) because when an application wants to read a character from a file, it would like to do it the same way whether the file resides on the floppy disk, the hard disk, or the CD-ROM; the operating system can provide a uniform "read a character" service, even though the different hardware devices each have different (and complicated) access mechanisms. Protection is necessary because an application program does not (usually) have carte blanche to operate all hardware devices: it can write to its own files, but is not allowed to write over the whole disk, is not allowed to display in other applications' windows on the screen, and so on.

27. The operating system *kernel* is a software component within the operating system that operates in "privileged mode" on the computer hardware, meaning that it has access to all of the computer hardware devices so that it can mediate services to applications. Examples of computer hardware devices include the video display, the disk drive, and the network communications hardware. *Device drivers* are software components installed within the operating-system kernel that allow the kernel to manipulate the individual hardware devices.

28. Most application software (*e.g.*, computer games, tax-return preparation software) is clearly not part of the operating system. However, some application programs that perform general utility functions (such as displaying the list of files in a folder) are often

included with an operating system and are conventionally called part of the operating system.

29. A useful rule of thumb to distinguish applications from kernel is that application programs execute in an unprivileged hardware mode (meaning they cannot directly access the computer hardware, but must rely on the operating system to provide hardware services), and the kernel executes in privileged mode (meaning it can directly control computer hardware). Device drivers run in privileged mode; the folder-display utility, word-processor, and tax software generally run in unprivileged mode.

30. Exhibit B illustrates the arrangement of software and hardware components on a typical computer. The components at the bottom are hardware devices, such as a mouse, keyboard, monitor, CD player, or hard drive. Above them are the device drivers (software components). These interoperate with the hardware devices through an *interface*, that is, a specification of what functions the devices can perform and how these functions are to be accessed by the drivers.

31. Above the device drivers is the operating system kernel. Each device driver interoperates with the kernel through a software interface called an application programming interface ("API"). The API specifies what functions each device driver can perform and how these functions are to be accessed by the kernel.

32. Above the kernel is an important layer of APIs (labeled 13 in Exhibit B), through which application programs interoperate with the operating system kernel. Like any interface, each of these APIs specifies what functions the kernel can perform, and how each application can access these functions. Some unprivileged (nonkernel) components of the

operating system (such as the folder-display utility application) also communicate with the kernel through these APIs.

33. An operating system vendor will usually publish, or disclose, these APIs so that application developers (also known as independent software vendors, or "ISVs") can make their programs interoperate with the operating system. Publishing an API provides the information that ISVs require to understand and make use of the functionalities available through that API. I will discuss the topic of disclosure further, in much greater detail, later in my testimony.

34. *Middleware* is application-level software that exposes one or more of its own APIs in order to provide services to other applications. Just as the kernel receives services from device drivers and repackages these to provide higher-level services to applications, middleware receives services from the kernel and repackages these to provide even higher-level services to its own applications. It is even possible that one of the clients of a middleware is itself middleware, providing services to even higher-level applications.

35. Exhibit C illustrates how this general picture of software components applies to the Microsoft Windows operating system product. The API layer between operating system and applications (labeled "13") is called the "Windows API" set. The Windows operating system product comprises the kernel (including some device drivers), certain utility applications, and some middleware (such as Internet Explorer). Other middleware and applications are independently provided by ISVs and need to interoperate with the Windows operating system kernel. Some middleware may even run on other middleware.

36. Some of the device drivers installed on a user's PC are built and distributed by independent hardware vendors ("IHVs") who make the hardware devices. These device drivers interoperate with the kernel through the API layer labeled as "15" on Exhibit C. In some cases, these IHV-created device drivers are distributed by Microsoft with its operating-system product; in others, the software is packaged with the device hardware (for installation by users) or is installed by OEMs.

37. The dashed lines inside box 14 of Exhibit C show internal interfaces (APIs) between software subcomponents. Within the kernel there may be thousands of subcomponents, and therefore thousands of APIs. These internal APIs are not generally visible to applications outside the kernel; the designers of the operating system choose which APIs are accessible by applications. In the diagram, this external API layer is labeled 13. Not only the kernel, but any substantial piece of middleware or application software will have many internal interfaces.

V. THE IMPORTANCE OF MIDDLEWARE

38. Middleware, described in basic terms, is software that exposes one or more of its own APIs to provide services to other software. In general, middleware is software that runs at the software application level (*i.e.*, runs as a client of the operating system or of other middleware) and provides services to other applications or other middleware. I believe that my understanding of the term "middleware" is consistent with that expressed in the District Court's Findings of Fact and the Court of Appeals decision in this case.

A. Ease of Portability

39. One of the primary benefits of middleware is that it can make software applications "portable." "Porting" software applications means to rewrite software code so that it is compatible with, and may operate on, a different computer system and/or operating system than the one for which it was originally developed. Based on my experience as a software developer**,** including my experience porting software and assisting others to port software that I have developed, I know that porting software applications can be a time-consuming and resource-intensive process.

40. Different operating systems provide different APIs that are based on different assumptions about the structuring of processes, file systems, and other operating-system resources. Application software based on one set of assumptions may need to be extensively rewritten to make use of an API that makes a different set of assumptions. The process of porting involves the analysis of these assumptions, the use of modularization to limit these assumptions to small portions of the application software, and the development of alternative versions of those portions to call upon a different operating-system API.

41. Different computer hardware architectures (*e.g.*, Intel Pentium, Sun Sparc) provide different "instruction sets," which encode the primitive step-by-step operations executed by the program. Porting is required to allow software written for one type of computer system to successfully run on another type of computer system. For instance, a program written for a computer system employing a Motorola chipset requires porting for it to successfully run on a personal computer system Intel x86 chipset. Assumptions about the

computer hardware can pervade an application program, which can make porting to a different hardware platform more difficult.

42. Applications coded to operate on middleware, however, require much less porting effort in order to operate on different operating systems or different computer systems when the middleware itself is ported. The middleware exposes the same APIs regardless of which operating system it is ported to. Component-software platforms such as the Java Runtime Environment expose the same virtual-machine instruction set to the application regardless of which hardware platform the middleware is ported to.

43. A single piece of middleware need not provide a complete set of APIs in order to help serve as an alternative platform for an application. Several pieces of middleware combined together can jointly serve as a platform for applications. For example, an application could use one middleware for an on-screen user interface, and (simultaneously) another middleware for playing sounds. Or, alternatively, it is possible that a single API may provide all of the functionality that an application needs to run.

B. Additional Advantages of Component-Software Middleware

44. Component-software middleware, such as Java and CLR, also has the particularly important benefit that applications written for these middlewares are less susceptible to viruses than most applications written to run directly on an operating system. This is because the applications that run on top of this “Java-style” middleware do not directly run native machine instructions. Applications running on such middleware are subject to translation by a just-in-time (“JIT”) compiler before they are executed. One of the tasks of the JIT compiler is to check programs to make sure that they will not perform

improper and unsafe operations, such as a buffer overrun. This checking feature of middleware reduces vulnerabilities to bugs and viruses.

45. These types of middleware also have the advantage of allowing for smoother interoperation of software components. Prior to the development of Java-style middleware, software developers faced two bad alternatives when choosing how to integrate different software components. One alternative was to separate the components so they communicated through the operating system. While this alternative protected each component from bugs in the other components and helped to preserve the stability of the application, it was a slow and inexpressive method for the components to communicate. The other alternative was to include the components within the same operating-system process. But this meant that a bug in one component could crash the entire application.

46. Java-style middleware allows software developers to avoid choosing between these poor options, because it allows the expressiveness and interoperability advantages of packaging components within the same process while reducing the risks that bugs in a single component will adversely affect the application.

C. Middleware Can Facilitate the Porting of Applications to Other Operating Systems

47. As discussed above, software products are able to be ported to other operating systems only with significant effort and expense. This makes it likely that a software product designed to operate on a particular operating system will not be ported to another operating system.

48. As I have explained above, middleware provides a means to make applications more easily portable: new applications developed to run atop middleware will be portable not by porting the application directly (which would be more work for the application developer) but by porting the middleware (which is done by the middleware developer). But middleware can also facilitate the porting of many existing applications that were not originally designed to run on the middleware. This is another effective path to allow a wide variety of ISV applications to be available on a variety of operating systems.

49. The applications that run on a particular software platform are written to that platform's particular set of APIs. Middleware can be built to support some of those same APIs, and the middleware itself can be ported to a number of different platforms. Thus, the applications can also be run on a variety of platforms without themselves having to be ported. I have illustrated this in Exhibit D, which shows a non-Microsoft operating system (such as Linux, MacOS, or Solaris) providing services through an API layer (labeled 13b) that is not compatible with the Microsoft operating system APIs. Running as an application atop this non-Microsoft API layer is a middleware that takes the services provided by the underlying operating system and repackages them in a form consistent with the Microsoft platform software APIs (labeled 7). Atop this middleware API layer are some of the same applications and middlewares that can run directly on the Microsoft operating system.

50. Of course, to develop this type of middleware, there must be sufficient disclosure of the APIs, Technical Information, and Communications Interfaces that are exposed by Microsoft platform software (both operating systems and middleware). Only in this way can third-party middleware support some of the same applications and middleware

products that currently run on Microsoft platform software. The subject of disclosure is discussed in the next portion of my testimony.

D. The Disclosure of Many Types of Information Is Necessary for Middleware Development

51. In order to develop software that can interoperate with a certain software platform, such as an operating system, software developers must have access to sufficient technical data – such as APIs and protocols – of that operating system to allow a program to utilize the functionality provided by the operating system.

52. An API is an interface provided by one software module to allow other modules to use the services it provides. An API consists of computer code that allows software to interact with (*i.e.*, call upon the features and functionality of) other computer code. In technical terms, an API is generally composed of functions, methods, classes, and data-structure definitions. These represent various ways that computer scientists can specify the exact functionalities available through an API.

53. In less technical terms, an API is analogous to the dashboard of a car: the dashboard is an interface, composed of wheels, knobs, pedals, and buttons, through which the engine compartment provides services to the driver. Just as sufficient documentation of an API is required in order to interface with a software module, sufficient specification of each dashboard control is needed for a driver to interoperate properly with a car. Just as with a dashboard, this documentation describes how to access functionality ("turn the steering wheel clockwise to turn the car right") and not how that functionality is implemented ("turning the steering wheel turns the collapsible shaft, which turns the pinion, moving the rack…").

54. Communications protocols specify the rules for the transmission of information between different devices. A protocol is often composed of several layers, with the lower layers providing simple, low-level services, and the upper layers repackaging the lower-layer services into higher-level services. For example, a low level service is "send this packet of information and hope it gets through;" a higher-level service is "send this packet and make sure it gets through uncorrupted." The higher-level service can be implemented by making use of the lower-level service. An even higher-level service is "send this long stream of messages in order," and so on.

55. A communications protocol specifies what messages, in what format, and with what meaning, are to be exchanged by the communicating machines. One machine may make services available to other machines, accessed through the protocol. Only software designers to whom the protocol is disclosed can write software that takes advantage of these services.

56. Without adequate disclosure of APIs and protocols (including adequate technical documentation of these items, described in more detail below), developers are unable to create software that effectively interoperates with the platform software.

VI. MICROSOFT'S REMEDY FAILS TO REQUIRE ADEQUATE DISCLOSURE OF APIs, PROTOCOLS, AND TECHNICAL INFORMATION

57. It is my opinion that Microsoft's Remedy does not require Microsoft to disclose sufficient APIs, technical information, protocols, and necessary documentation to allow the development by third parties of platform software and applications that can interoperate with Microsoft platform software and applications. It is my further opinion that the disclosure provisions of the States' Remedy are necessary and adequate to allow

the development of such non-Microsoft middleware and other platform software products. The following are the reasons and bases for these opinions.

A. Microsoft's Remedy Fails to Require Microsoft to Disclose Necessary Technical Information

58. Although Microsoft's Remedy requires Microsoft to disclose certain (but not all) APIs and communications protocols exposed by its operating system products, it does not appear to require disclosure of the significant amount of other technical information that is necessary to achieve proper functionality and interoperability between the operating system product and software that runs on it.

59. Operating-system APIs provide the middleware or application with the information necessary for it to make the proper "calls" to receive services from an operating system. Disclosure of the APIs allows software developers to write code that can properly make these "calls." But the APIs do not provide the means for interpreting the data that is received from the operating system. In order to interpret, and thus make use of, the data received from the operating system through the API, the middleware or application must make use of additional information, such as data structure definitions and layouts, syntaxes and grammar. Furthermore, threading and synchronization conventions, as well as memory allocation and deallocation conventions, prescribe the order in which different calls should be performed (by analogy, a driver unfamiliar with the protocol that "you must step on the brake while shifting out of Park" would be unable to operate cars manufactured since 1995).

60. I have reviewed Section III.D of Microsoft's Proposed Remedy, and the relevant definitions, and it does not appear that Microsoft would be required to disclose much of

this additional technical information that is required for software to interoperate effectively with the Microsoft operating system products. For example, Section III.D obliges Microsoft to disclose "the APIs and Related Documentation that are used by Microsoft Middleware to interoperate with a Windows Operating System Product." But Microsoft's definition of "API" is almost entirely a circular reference to Section III.D, and it therefore does not adequately define what information must be provided as part of the API disclosure. *See* Section VI.A. And Microsoft's definition of "Documentation" ties its disclosure requirements to "the sort and to the level of specificity, precision and detail that Microsoft customarily provides for APIs it documents in the Microsoft Developer Network." *See* Section VI.E. Nothing in Section III.D appears to prevent Microsoft from disclosing to its own middleware developers a level of specificity and detail far beyond what it customarily provides through MSDN to developers of non-Microsoft Middleware. There is no requirement to disclose the additional technical information that is needed by ISVs to create competing software, for example, middleware that supports the applications that run on Microsoft middleware. In other words, nothing in Section III.D appears to require Microsoft to disclose to any degree of technical specificity beyond what Microsoft discloses today.

61. I have the same concern with respect to Section III.E. Indeed, Section III.E appears to impose no obligation to disclose technical data needed to use a particular Communications Protocol in an effective manner.

62. It also appears that Microsoft's Remedy does not require Microsoft to disclose another important category of information, namely, file formats. Microsoft applications, and especially the Microsoft Office middleware, create files that are exchanged with other

users and other applications – for example, Word documents, Excel spreadsheets, and PowerPoint presentations. I often receive Word documents and Excel spreadsheets in e-mails from colleagues elsewhere. In order to view and edit these documents, I need word-processor or spreadsheet software. The software must be able to open and parse the document.

63. A Word document is not just the text that the human reads, but includes encoded markings indicating which text is in italics, where the font changes, how the paragraphs are separated, and other document-structuring information. If these encodings are not disclosed and documented, then it will be difficult or impossible for ISVs to make word-processor software that can process these documents. That is, an important form of interoperation between Microsoft middleware and non-Microsoft software is, or could be, through the exchange of files in these file formats. But the Microsoft Remedy fails to require disclosure of the information needed for interoperability with Office file formats.

64. I understand that the States' Remedy would require the disclosure of the types of technical information discussed in the preceding paragraphs. I understand that Section 4.a of the States' Remedy would require the disclosure of "APIs, Technical Information and Communications Interfaces" with respect to Microsoft Platform Software. The definition of "Technical Information" set forth at Section 22.nn of the States' Remedy includes important information that appears not to be addressed by Microsoft's Proposed Remedy, including file formats, data formats, conventions, syntaxes, and data layouts. It is my opinion that this broader disclosure of technical information set forth in the States' Remedy is necessary to allow independent software developers to create programs that

effectively interoperate with Microsoft platform software, including the Windows operating system products.

B. Microsoft's Remedy Does Not Require Disclosure of the APIs Exposed by Microsoft Middleware

65. Under Section III.D of Microsoft's Remedy, only "the APIs and related Documentation that are used by Microsoft Middleware to interoperate with a Windows Operating System Product" must be disclosed. This excludes an important category of interfaces needed for interoperability, namely, the interfaces between Microsoft middleware and applications.

66. Disclosing middleware interfaces is important because there are a number of important interfaces that are exposed by Microsoft's middleware rather than by the operating system. For example, Microsoft's Internet Explorer browser exposes an important API that is used by the Java middleware platform to run on the PC. Microsoft does not currently publish this API, so third-party suppliers of Java cannot guarantee that their Java platforms will work properly on Windows PCs. *See* Trial Testimony of Richard Green at 364-66 (Mar. 19, 2002).

67. These disclosures are particularly important in the case of Microsoft middleware products such as Internet Explorer and Windows Media Player that are distributed with the Windows Operating System Product. Software is infinitely malleable, and functionality can be moved from one software component to another in the discretion of the developer. Thus, Microsoft could choose to move its most significant interfaces from the operating system level to the middleware level. If the Remedy does not require disclosure of interfaces exposed by Microsoft middleware, then important interfaces will be permitted to remain undisclosed.

68. The result would be that information necessary for the development of applications to interoperate with Microsoft middleware would not be required to be disclosed. This allows Microsoft the discretion over what functionalities it may provide to applications. It may be the case that a Microsoft middleware product supports both Microsoft applications and non-Microsoft applications, in which case it is possible for Microsoft to provide information on particularly advantageous functionalities internally to its own application development team but not externally to other application developers.

69. The States' Remedy would require the disclosure of interfaces exposed by Microsoft middleware. Section 4.a.i of the States' Remedy requires the disclosure of "all APIs, Technical Information and Communications Interfaces that Microsoft employs to enable . . . each Microsoft application to Interoperate with Microsoft Platform Software installed on the same Personal Computer." Section 4.a also makes clear that the purpose of its disclosure provisions are to enable "non-Microsoft Platform Software and non-Microsoft applications to Interoperate with Microsoft Platform Software and/or applications for Microsoft Platform software." It is my opinion, therefore, that the States' Remedy addresses this important category of interoperability disclosures; a category excluded from the disclosure provisions of Microsoft's Remedy.

C. Microsoft's Remedy Contains an Overly Restrictive Definition of Middleware That Excludes Many Necessary Disclosures of APIs and Other Technical Information

70. Section III.D of Microsoft's Proposed Remedy provides for disclosure of "the APIs and related Documentation that are used by Microsoft Middleware to interoperate with a Windows Operating System Product." Section IV.J of Microsoft's Proposed Remedy provides a definition of "Microsoft Middleware" that is overly restrictive and fails to

include a number of important middleware products. In combination, these provisions of Microsoft's Proposed Remedy would appear to omit from the API disclosure requirements significant numbers and types of APIs needed by third parties to develop software, including middleware, that can properly interoperate with Microsoft platform software.

71. The result of this omission is that developers would not have access to APIs exposed by the Microsoft operating system that are not used by the specific products that satisfy Microsoft's restrictive definition of "Microsoft Middleware." Because Microsoft can decide which operating-system APIs its own middleware uses, this gives Microsoft complete discretion to control which APIs are disclosed under the terms of Microsoft's Remedy. Microsoft could therefore remove important APIs from its disclosure requirements. As a result, developers would not be able to develop middleware or applications that effectively make use of these particular APIs.

72. Unlike Microsoft's Remedy, Section 22.x of the States' Remedy contains a more comprehensive and appropriate definition of "Microsoft Middleware Product." It is my opinion that each example and/or description of software provided in this definition, and in the definition of "Middleware" incorporated therein, meets the technical definition of "middleware." I understand that Section 4.a of the States' Remedy requires Microsoft to disclose "APIs, Technical Information, and Communications Interfaces" used by "each Microsoft Middleware Product to Interoperate with Microsoft Platform Software." Thus, it is my opinion that the States' Remedy would provide the broader level of disclosure of technical data necessary for independent software developers to create software that effectively interoperates with Microsoft platform software.

73. The following are only two examples of important middleware developed by Microsoft that would appear to be excluded from the definition of "Microsoft Middleware" under Microsoft's Remedy (and thus also from the disclosure requirements of Microsoft's Remedy) but that would be included in the disclosure provisions of the States' Remedy: CLR and Office.

1. Common Language Runtime ("CLR")

74. Microsoft's .Net CLR is an example of component-software-platform middleware. It is the functional equivalent of Java. In particular, both CLR and Java provide middleware platforms on which other software applications can run. Both CLR and Java define "virtual machine language" that allows applications to be independent of the hardware platform's instruction set. Both CLR and Java contain a "verifier" to inspect applications for security problems before running them, and a "just-in-time compiler" to translate application programs from the virtual machine language into the hardware's native machine code. Both CLR and Java provide automatic memory management ("garbage collection") services to applications. Even at a level of greater technical detail there are many structural similarities between Java and CLR. The C# programming language, introduced by Microsoft for programming CLR, bears a great resemblance to the Java programming language, introduced by Sun for programming Java systems.

75. Unlike Java, however, CLR is a Microsoft product that was not developed to be ported to other operating systems. Thus, applications developed for CLR will only run effectively on Windows, and they may not work at all on competing operating systems.

76. It appears that CLR may not be included within Microsoft's Proposed Remedy's definition of "Microsoft Middleware." I have reviewed the deposition transcript of James Allchin, Microsoft Group Vice President, Platforms Group, who testified that CLR does not meet the definition of "Microsoft Middleware" under Microsoft's Remedy.[5] And it is not clear that CLR would be considered to be distributed separately from a Windows Operating System Product or to provide the same or substantially similar functionality as a "Microsoft Middleware Product," as would be required to fit the definition of "Microsoft Middleware." *See* Section IV.J, K. Thus, Microsoft's Remedy does not appear to require Microsoft to disclose the APIs that are used to allow CLR to interoperate with the Microsoft operating system. Nor would it require the disclosure of the APIs that CLR makes available to other applications (because, as discussed above, Microsoft's Remedy does not require disclosure of the APIs exposed by Microsoft Middleware).

77. It should be technically possible for ISVs to make middleware that provides the same functionality as the Microsoft CLR middleware by supporting those same applications through a compatible interface; this interface is called the Common Language Interface

[5]At his deposition Mr. Allchin stated:

> Q. Turning again to the RPFJ, which is Allchin Exhibit 2, and specifically to definition J for Microsoft middleware. Can you tell me whether CLR is included in the RPFJ's definition of Microsoft middleware?
>
> A. … I would say that it does not – does not fit this definition today, although we are planning on treating it that way.

(Allchin Dep. Tr., Feb. 13, 2002 at 86). Microsoft's apparent decision, in its discretion, possibly to treat CLR as middleware does not change my technical opinion that the terms of Microsoft's Remedy do not appear to include CLR within the definition of "Microsoft Middleware."

(CLI). If the third-party middleware were to run on top of competing operating systems (other than Microsoft operating systems) then this would make these applications portable to the other operating system(s). However, it is my opinion that non-Microsoft versions of CLR-style middleware cannot offer full interoperability without two forms of fuller disclosure (neither of which is required by Microsoft's Remedy).

78. First, information relating to the services, such as the APIs and protocols, that CLR utilizes from the Microsoft operating system must be fully disclosed and documented. This information is necessary to allow third parties to develop middleware that effectively uses these APIs and protocols, and thus effectively interoperates with the Microsoft operating system.

79. Second, it is important that information relating to the services, such as APIs and protocols, that CLR makes available to applications (written to run upon it) is disclosed and documented. This information is necessary to allow third parties to develop middleware that can properly support applications that are written to run on CLR. This creates the opportunity for third parties to develop cross-platform middleware that allows for the portability of existing applications to non-Microsoft operating systems.

80. Unlike Microsoft's Remedy, the States' Remedy appropriately includes CLR within its definition of "Middleware" and "Middleware Product." *See* Sections 22.w and 22.x.i. As a result, under Section 4.a, the States' Remedy requires the disclosure of "APIs, Technical Information, and Communications Interfaces" between CLR and applications and between CLR and Microsoft platform software. It is my opinion that such disclosure is necessary to allow independent software developers to create software, in particular

middleware, that interoperates with Windows using the APIs and other technical data used by CLR and that supports applications that have been developed for CLR.

2. Microsoft Office

81. Microsoft Office is also an important middleware platform that is not included within the definition of "Microsoft Middleware" under Microsoft's Remedy. Microsoft Office is a suite of business productivity software, including Word, Excel, Outlook, and PowerPoint. Microsoft Office serves as one of the most important middleware platforms in the current business world, exposing APIs that allow other software to operate on it. The structure of modern business applications often consist of custom programming, written in languages such as Visual Basic, that invoke multiple features of Office components, such as Word, Excel, and Access, in order that data can be properly retrieved and displayed in a familiar way. There are entire books written on how to use Office as a platform. *See, e.g.,* Peter G. Aitken, *Developing Solutions with Office 2000 Components and VBA* (Prentice Hall PTR 2000).

82. A business application typically requires databases for storage of information, business logic to decide what to do with the information, and user interfaces to allow users to manipulate the information as permitted by the business logic. The information might be a set of purchase orders; the business logic might control the routing of purchase orders within a company and the authorization of certain individuals to approve purchase orders; and the user interface would display purchase-order forms and allow users to fill in the blanks. The business logic might be written as an application in the Visual Basic programming language, which runs on top of the Microsoft Office platform and which

uses the Access component of Office for data storage and the Word component for displaying forms.

83. Thus, applications written for this middleware enjoy consistency of data storage and user interface with other such applications. Developers of applications that run atop Office can focus on writing their own business logic, and let Office do the manipulation and presentation of documents, forms, spreadsheets, and databases. Based on my review of textbooks on developing applications for Office and on consultation with a developer of commercial business software,[6] I know that Office serves as an important platform for application development.

84. The file formats of Office represent another important aspect of the Office platform. As discussed above, Office file formats, such as .doc, .xls, and .ppt, are examples of the type of information necessary for interoperability between Office running on one PC and Office running on another PC (in instances in which documents are exchanged over a network) or between Office and non-Microsoft business productivity software (running on the same PC or different PCs).

85. Microsoft's Remedy appears not to include Microsoft Office and its component software within the definitions of "Microsoft Middleware" or "Microsoft Operating System Product." Section IV.J, K. Thus, Microsoft's Remedy does not appear to require it to disclose the APIs (or other technical data) that are used to allow Office to interoperate with the Microsoft operating system or that Office makes available to other applications.

[6] I discussed this subject with Steve Sashihara, President of Princeton Consultants, Inc., who has developed applications that run on the Microsoft Office platform.

Without such disclosure, independent software developers cannot develop software that interoperates with the APIs and file formats used by Office, or that interact with other applications in the manner that Office interacts with those applications.

86. For example, an ISV may wish to develop an application product that competes directly with one component of Microsoft Office (*e.g.*, the Powerpoint presentation software) while interoperating with other Office components. An ISV might do this because it lacks the resources to simultaneously develop an entire suite of office applications, or because many customers are already familiar with some of Microsoft's office components. In order for a competing component to interoperate with Microsoft's Office components, the interfaces and file formats must be documented.

87. Unlike Microsoft's Remedy, the States' Remedy appropriately addresses the important middleware aspects of Office by including it in its definition of "Middleware" and "Microsoft Middleware Product." *See* Sections 22.w and 22.x.i. Section 22.nn also includes file formats in the definition of "Technical Information." Section 4.a of States' Remedy would therefore require Microsoft to disclose "all APIs, Technical Information and Communications Interfaces" relating to the interoperability of Office. It is my opinion that these disclosure requirements are necessary to allow interoperability with Office, an important middleware platform.

D. The Disclosure Requirements of Microsoft's Remedy Apply Only to a Far Too Limited Purpose

88. The disclosure requirements of Microsoft's Remedy are also inadequate to allow for the development of non-Microsoft platform software because Microsoft's Remedy mandates disclosure only for a very limited purpose.

89. Section III.D of Microsoft's Remedy requires Microsoft to disclose APIs only "for the sole purpose of interoperating with a Windows Operating System Product." Similarly, Section III.E provides that Microsoft shall disclose Communications Protocols only "for the sole purpose of interoperating with a Windows Operating System Product."

90. These restrictions would inhibit certain types of software development. For example, if a software developer wanted to develop a non-Microsoft operating system, the developer would reasonably want its operating system to support at least some of the same applications and middleware that run on the Microsoft operating system. As explained above, the cost and burden of porting applications deters ISVs from rewriting their applications to run on the new operating system. Thus, the developer of the new operating system may want to create interfaces that will support preexisting applications that have been developed for other operating systems (in particular Windows, which is the dominant PC operating system product). In order to do so, the developer would need access to the APIs that the Microsoft operating system uses to interact with applications and middleware.

91. Similarly, if the Microsoft operating system or other Microsoft platform software communicates with a client (or server) through a communications protocol, a non-Microsoft operating system (or software product) would need to communicate with the same client (or server) using the same protocol. In such situations, the developer would seek access to those protocols not "for the sole purpose of interoperating with a Windows Operating System Product," but rather in order to provide an alternative to a Windows Operating System Product.

92. Sections III.D and III.E of Microsoft's Remedy appear not to require disclosure of Microsoft's APIs and protocols to such a developer for this purpose. In contrast, I understand that the disclosure requirements provided by Section 4 of the States' Remedy do not contain any similar limitation concerning the purpose for which the disclosed information would be used. In fact, Section 4 clearly states that the purpose of the disclosures includes enabling non-Microsoft platform software to interoperate with Microsoft middleware and applications for Microsoft platform software.

E. Microsoft's Remedy Does Not Allow Access to Microsoft Source Code, Which Is Often Necessary for Software Development

93. Although disclosure and documentation of APIs, Technical Information and Communications Interfaces is necessary and important, there may be instances in which further information is needed by developers in order to make use of such information. In an ideal world, if documentation of an API is full, clear, and technically sophisticated, there is no need for the developer of one component to read the source code of the component on the other side of the API. Thus, disclosure of an API does not usually entail disclosure of the source code behind that API. However, producing such technically sophisticated documentation is not always easy, and in my experience as a software developer, I have often found API documentation to be inadequate for purposes of developing software that must interoperate with it. In such cases, the decades-old practice in software development is, "when in doubt, read the source;" that is, one can gain better understanding of an interface by reading the source code on the other side of the interface.

94. There are several circumstances in which the documentation of an API is not entirely adequate to allow software developers to make use of the interface. In some instances, this is due to the fact that the interface is particularly complex and difficult to document. In other instances, it is due to the fact that the software being developed to interact with the interface is innovative, and its functions were not envisioned when the interface was documented. And particularly with respect to such innovative uses, it can sometimes be the case that use of the API as documented may result in unintended consequences elsewhere in the platform software. In such cases, reviewing the source code of, for example, a platform software, is invaluable for ensuring proper interoperability with that platform software.

95. It appears that Microsoft recognizes this important form of disclosure as a means of assisting the development of software. Brian Valentine, Senior Vice President of Microsoft's Windows Division, testified at his deposition that Microsoft has a program to make source code available to certain ISVs to help them in developing software.[7] Similarly, the testimony of Steven McGeady, a former executive with Intel, demonstrates

[7] Mr. Valentine testified:

> Q. And do the ISVs and others have access to the source code directly?
>
> A. We have a source code access program that we can deliver to corporate customers, certain partners, OEMs, ISVs can get access to the source code, but it's also documented clearly in there that you should not be using internal APIs, it should be more if you need to debug something in a complex debug environment, having access to the source might help you debug something and those kinds of things, but we are very against people using internal APIs.

(Valentine Dep. Tr., Jan. 29, 2002 at 47-48).

that interface disclosures can be insufficiently documented and how, in at least one instance, Intel engineers were able to achieve interoperability only after Microsoft provided them with access to the relevant source code.[8]

96. Microsoft's Remedy provides no opportunity for software developers to review the source code of Microsoft platform software. This can be a significant hindrance to the development of compatible software, particularly when Microsoft fails to provide, or is unable to provide, adequate documentation as required under the other provisions a remedy or fails to do so in a timely manner.

97. In contrast, I understand that Section 4.c of the States' Remedy would allow software developers and other parties to study and otherwise make effective use of such Microsoft software source code, while restricting their use of this source code for the purpose of interoperating with Microsoft platform software. It is my opinion that this remedy provision is necessary to allow such third parties to develop products that can effectively interoperate with this Microsoft software.

98. Furthermore, an important purpose of Section 4.c is "monitoring of compliance." If an ISV who makes a non-Microsoft Middleware has reason to believe that Microsoft Middleware is taking advantage of undisclosed Microsoft APIs, inspecting the source code may be one of the only ways to independently determine whether Microsoft is complying with the Remedy. This is true not only of the disclosures required by the States' Remedy in Sections 4.a and 4.b, but even of the disclosures required by Microsoft's Remedy in Sections III.D and III.E.

[8] *See* Steven McGeady Direct Testimony at ¶¶ 21, 30, 39.

VII. THE DISCLOSURES REQUIRED BY THE STATES' REMEDY WOULD NOT ALLOW THE COPYING OF WINDOWS

99. In my depositions, Microsoft questioned me about whether Section 4 of the States' Remedy would allow the "cloning" of Windows. As a technical matter, I do not believe that the term "cloning" is a precise technical term, and to avoid unnecessary confusion, I will not use it in addressing this issue. To the degree such questions are meant to suggest that the States' proposed disclosure provisions would provide technical information that would allow someone to make a copy of Windows (or other Microsoft platform software), it is my opinion, based on a careful review of its disclosure terms, that the States' Remedy does not allow such copying.

100. In order to encourage the development of software to run on its platform software, Microsoft already discloses many of its interfaces. But the remedies submitted by both Microsoft and the States recognize that these current disclosures are insufficient, and they both include provisions requiring more complete disclosures. As I discussed previously, Microsoft's Remedy has many shortcomings in its disclosure provisions, which provides Microsoft discretion to determine exactly who receives certain disclosures necessary to interoperate. The States' Remedy provides a more comprehensive description of the disclosures that Microsoft must make in order to allow interoperability with Microsoft platform software and the applications that run on those platforms. These provisions require the disclosure of interfaces between software, not of the underlying source code of the software itself. In my opinion, these disclosures are not the quantity or type that would allow for the copying of Windows.

101. Both the internals of an operating system (such as the contents of box 14 in Exhibit C) and the APIs between components (such as the Windows API set located at line 13) are written in programming languages such as "C" or "C++"; this is generically called "source code." Each component is built from "functions;" a function might be a few lines of source code, or even hundreds of lines. Each function calls upon other functions to help it in its task; these other functions might be in the same component or they might be in other components. Each component exposes only some of its functions for use by other components; the API for a component lists which functions are available for other components to use, and lists exactly how to call upon each function. A function not listed in an API is "internal," meaning that it is available for use only by other functions within the same component.

102. I have shown in Exhibit E an example of components and the APIs between them. I have written a sample "application" (in the file client.c) that prints all the prime numbers between 1 and 100, atop a "platform software" that tests whether a given number is prime. Between them is an API (located in the file testprime.h) that exposes one function, called is_prime. The platform software itself is composed of two components (in the files testprime.c and divides.c) with an internal API between them. The source code in testprime.h is the *interface* between components; the source code in testprime.c and divides.c is the *implementation* that supports that interface. I can describe *what* the platform software does ("test whether a number is prime") without describing *how* it does it; similarly, I can disclose the API (the one-line file testprime.h) without disclosing the full source code for the implementation (the 17 lines of testprime.c and divides.c).

103. However, to make the API disclosure technically useful, more than just the one line of the testprime.h file must be disclosed; that file tells the name of the function, while leaving the application developer guessing about the function's purpose. One should also provide documentation that explains how to use the API. For example, such documentation might say, "is_prime(n) returns TRUE if n is prime." A convention for using this API, such as "it is only guaranteed to be accurate if n is greater than 1," would also need to be disclosed to properly use the API. However, such documentation is not a disclosure of the source code; it describes in greater depth *what* the function does, not necessarily *how* the function does its work.[9]

104. A large software component such as an operating system kernel is typically built from many smaller components. These components expose many APIs to each other — these are APIs internal to the operating system — but only a small subset of these APIs are exposed outside of the operating system. In the case of Microsoft Windows, these externally exposed APIs constitute the set of "Windows APIs," shown as line 13 of Exhibit C.

105. Based on my experience as a software developer, I am of the opinion that it would be impossible to make a copy of an operating system based solely on the disclosure of APIs. This is true simply because the APIs constitute only a very small percentage of the total operating system source code. Thus, someone wishing to copy the operating system would still need to independently develop and test the vast majority of source code that

[9] I have constructed an extremely simple API and implementation for this example and therefore only limited disclosures would be necessary to document the API adequately. Depending on the functionalities and complexity of a given API, different types and depth of documentation will be necessary. *See supra* Section VI.A.

implements the disclosed APIs. Even if the person could develop such a different operating system that exposes the same APIs, the task would be monumental.

106. To confirm this opinion based on my own software development experience, I have directed and supervised a review of the Microsoft source code for its current Windows Operating System Product, Windows XP.[10] I have found that Microsoft's source code constitutes at least 38 million lines of code.[11] Of this, approximately 440,000 lines,[12] or 1.2%[13] of the source code, constitutes the APIs that Microsoft currently discloses to application developers through its "MSDN" web site and its Visual Studio development tools. This confirms my view that the APIs represent only a very small percentage of the total Windows source code.

107. I also note that in many instances, the disclosure that would be required under the States' Remedy would not be of additional APIs (and their source code). Rather, in many instances, the required disclosures would be of the Technical Information needed by third

[10] The procedures for and summary of this review are attached as Exhibit F.

[11] I understand that this source code was first made available for my review in this case on or about February 20, 2002. Because of the limited time available in making these measurements before trial, I was not able to obtain exact results. There were approximately 8 million lines that I was unable to classify definitively as either source code or not, so the true figure might be as high as 46 million lines of source code. The review of the source code is ongoing, and I may be able to provide more accurate calculations after this further review.

[12] These lines of code represented the disclosed Windows APIs and were determined from the "include files" provided with the Microsoft Visual Studio development tool. The include files provide the source code for the disclosed Windows APIs.

[13] I conservatively estimate the error in these measurements to be 20% to 40%. Thus, for example, the true measure of how much of its source code Microsoft discloses through published APIs (relative to the size of its operating system implementation) might be as little as 0.85% or as great as 1.7%.

parties to make effective use of the APIs. Such disclosures – while extremely important for purposes of achieving interoperability – would not result in the disclosure of additional Microsoft source code. Rather, they would only mean that disclosed APIs are explained more fully so that developers properly understand how APIs work and can write software products that interoperate effectively with Microsoft platform software. None of the examples of Technical Information disclosed under the States' Remedy require the disclosure of any source code (except for reference implementations, discussed below, which are merely sample software programs and not the actual source for the operating system.) Thus, these types of disclosures would not require the publication of additional Windows source code.

A. Reference Implementations

108. The States' Remedy (Section 22.nn) describes many kinds of "Technical Information" that might be necessary to adequately document an API or Communications Interface. Whether any particular type of technical information needs to be disclosed is governed by Section 4, which requires disclosure of the "APIs, Technical Information and Communication Interfaces" needed to enable certain categories of software to interoperate. Different interfaces require the disclosure of different kinds of technical information; no single interface is likely to require *all* the kinds of information enumerated in the last sentence of Definition 22.nn. It is, however, my opinion that all of the types of information included in Definition 22.nn would be necessary forms of disclosures to allow at least some software to interoperate with Microsoft platform software and applications for that software.

109. Microsoft has questioned me about the inclusion of "reference implementations" in Definition 22.nn. I will attempt to explain what a reference implementation is. One way to explain the purpose of a function in an API is to use ordinary English, or English mixed with mathematics. For example, in the program in Exhibit E, I might say "is_prime tests whether a number is prime; a prime number is one that is divisible only by itself and 1." But sometimes it is difficult to explain a concept in a sufficiently precise way using English. In such a case, another way to explain what a function does is to provide simplified source code of equivalent functionality. Such source code, called a "reference implementation," is not meant to be the same as the source code that's actually used in the real implementation. The reference implementation is meant to be easy to understand for the human; the real implementation is meant to be as efficient as possible. The two implementations may be quite different. The disclosure of a reference implementation for some component does not necessarily disclose the actual source code of that component. Thus, the inclusion of the term "reference implementation" in Definition 22.nn does not mean that Microsoft would be required to publish the source code of its operating system.

110. Not every API requires a reference implementation. Most APIs can be documented to adequate technical depth without the use of a reference implementation. However, for those that cannot be adequately documented by other means, a reference implementation may be an appropriate form of disclosure.

B. The Secure Facility for Examining Source Code

111. The States' Remedy (Section 4.c) provides for "a secure facility where qualified representatives . . . shall be permitted reasonable access to study . . . the source code . . .

of Microsoft Platform Software." Software developers may need to resort to a review of source code when the documentation of an API is technically inadequate to explain what functionality the API provides; in such a case inspection of the source code of the implementation may be the only effective means to achieve interoperability.

112. In my opinion, this provision would not allow the copying of Microsoft source code. As an initial matter, it is not necessary to examine the entire source code of a product in order to achieve interoperability with it. It is generally sufficient to examine only the relevant portions of the source code that would be utilized by the interoperating software.

113. Even if the entire Microsoft source code were made available for examination, however, copying it would be an exceedingly difficult task. For example, as previously noted, the Microsoft Windows XP source code consists of at least 38 million lines of code. My understanding is that the States' Remedy would allow appropriate persons to examine this source code for the limited purpose of achieving interoperation, but not to make a copy of the source code while in the secure facility. As a result, to make a copy of the Windows source code, the examiner would have to memorize all of the lines of code – a virtually impossible task given the size of the Windows code (even if the examiner had many assistants in such an effort).

114. Moreover, although I am not an expert in software licensing issues, I note that Section 4 of the States' Remedy places a restriction on the purpose for which persons can gain access to the secure facility, namely, "for the purpose of enabling their products to Interoperate effectively with Microsoft Platform Software." I also note that Section 15 of the States' Remedy includes a provision stating that "nothing herein shall require

Microsoft to permit the use or sub-licensing of any Microsoft source code beyond that necessary to permit the exercise of such options or alternatives" available under the States' Remedy. I believe that this means that Microsoft could place reasonable restrictions on persons seeking to examine its source code. For example, Microsoft could require such persons to sign licenses requiring them to review the source code solely for the purpose of creating interoperable software products and not to disclose publicly the information that they learn. Such licensing restrictions would prohibit the copying of Windows and other Microsoft products.

VIII. MICROSOFT'S REMEDY PROVIDES AN OVERLY BROAD AND TECHNICALLY UNJUSTIFIED SECURITY EXEMPTION THAT COULD SIGNIFICANTLY RESTRICT TECHNICAL DISCLOSURE BY MICROSOFT

115. As described previously, computer security is one of my particular research interests and areas of expertise. For example, I have published papers on distributed authentication protocols, on security of virtual machines (such as Java), and on programming-language features to support security. I regularly read the scientific literature on both the practical and the theoretical aspects of cryptography and security, and participate in discussions of these issues with other scientists. In 2001, I was selected to serve on the Program Committee of the 2002 IEEE Symposium on Security and Privacy, a well-known annual conference on computer security.

116. Section III.J.1 of Microsoft's Remedy provides an exemption from the disclosure requirements imposed under the other provisions of Microsoft's Remedy, ostensibly based on Microsoft's need for maintaining security. Section III.J.1 exempts Microsoft from disclosing "portions of APIs or Documentation or portions or layers of Communications Protocols the disclosure of which would compromise the security of a

particular installation or group of installations of anti-piracy, anti-virus, software licensing, digital rights management, encryption or authentication systems, including without limitation, keys, authorization tokens or enforcement criteria." It is my opinion that this provision is premised on a concept of computer security that is inherently flawed as a matter of computer science and that is overly broad for any legitimate purpose.

A. The Disclosure Exemption of Section III.J.1 Is Not Technically Justified

117. There is no technical justification for the broad security exemption of Section III.J.1. The security of properly designed software should not be jeopardized by the disclosure of APIs and other technical information.

118. Software often must interact with an operating system in secure ways. This security is achieved through the implementation of a set of APIs and communication protocols, sometimes in combination with authentication keys or tokens. The APIs and communication protocols provide the security algorithm, which describes the method that must be followed when communicating secure information. An authentication key or token provides the information necessary to verify that an individual has the proper authority for secure access. The key contains a small piece of digital information that is used as input into the security algorithm to scramble the information in a unique way. The key, and the digital information contained therein, are not publicly disclosed. If the secrecy of the key is compromised, the key is simply deactivated and replaced.

119. The idea of providing security by keeping the APIs and communication protocols a secret is often described as "security through obscurity," and it is considered a well-known fallacy of security in the field of computer science. The testimony of Steven McGeady

highlights that the fallacy of "security through obscurity" is widely recognized by those dealing with security issues in leading companies in the computer industry, such as Intel.[14]

120. It is well understood in the field of computer science that security is not achieved through the nondisclosure of algorithms, but through the issuance of private keys, for several reasons: ease of response to compromised secrets, scientific review, and interoperability. I will explain each of these.

121. Security through obscurity not only needlessly causes interoperability issues for software attempting to communicate securely, it does not provide effective security. In a secure environment, all users employ the same algorithm embodied in the APIs and communication protocols. If the security of this environment depends upon keeping the APIs and communications protocols secret, this security is compromised – indeed, lost – if the secret information ever becomes publicly available through, for example, reverse-engineering, hacking, or internal leaks. Further, a great deal of time is spent designing the security APIs and communication protocols for a security algorithm and they cannot be easily rewritten when their secrecy is lost. Security that resides in an individual key or token does not threaten the security of the algorithm if disclosed. Further, the loss of an individual key does not require the underlying code for the security algorithm to be changed.

122. The best and most prevalent method for achieving secure communication is public disclosure of the APIs and communication protocols underlying a security algorithm in

[14] *See* Steven McGeady Direct Testimony ¶¶ 56-78.

conjunction with the issuance of private authentication keys. It is widely accepted that public review of security algorithms leads to better security: scientists and practitioners have an opportunity to analyze the algorithm or protocol and comment on its weaknesses. Secure Sockets Layer (SSL) and Kerberos represent widely adopted industry security standards that employ this method of achieving security. On the other hand, there are many examples of such secret, non-peer-reviewed security algorithms (such as Contents Scramble System, or "CSS") that were deployed and later found to have serious weaknesses.

123. Public disclosure of the APIs and communications protocols allows for interoperability between software employing the same security scheme. Nondisclosure of the APIs would needlessly create interoperability problems for individuals attempting to interact with Microsoft Code through non-Microsoft software. Despite the fact that users may properly obtain a security key, the key becomes useless if software developers cannot write their programs to process the key to correctly interoperate with the Microsoft code in the manner required by the security algorithm.

124. The nondisclosure of APIs does not automatically make them secure. As a technical matter, anyone who purchases a copy of the Microsoft operating system can painstakingly "reverse engineer" it to see its inner workings. This is true whether or not such reverse engineering happens to be legal in the state or country where it occurs. Thus, even if Microsoft were to avoid disclosing APIs that would lead to "the location of [cryptographic] keys," such locations can be found. In fact, shortly after Microsoft released Windows XP in the fall of 2001, someone reverse engineered the product and located undisclosed APIs, thereby discovering the location of cryptographic keys. The

person then published an analysis of his accomplishment on the Internet under a pseudonym. *See* “Microsoft’s Digital Rights Management Scheme – Technical Details” by Beale Screamer (pseud.), published October 18, 2001 at cryptome.org.

125. I also note that Dr. Roger Needham, the director of Microsoft’s Cambridge Research Laboratory, was questioned in his deposition about the security exemption of Section III.J.1. He testified that the scope of disclosure exemptions necessary to promote security was much narrower than the broad range of disclosure exemptions listed in Section III.J.1. Specifically, he testified that nondisclosure was necessary only with respect to APIs whose disclosure would result in discovery of [cryptographic] “keys or the location of keys.”[15]

B. The Disclosure Exemption of Section III.J.1 Is Overly Broad

126. Section III.J.1 is also overly broad and would allow non-disclosure of APIs and other technical information in many instances that do not further any legitimate security purposes. For example, Section III.J.1 allows Microsoft to avoid disclosure if Microsoft (apparently in its discretion) considers the disclosure to compromise the security of digital rights management systems (“DRM”). Some new media formats have DRM content protection mechanisms that prevent the playing of media by unlicensed devices (such as media players). For example, Section III.J.1 could be interpreted by Microsoft

[15] Deposition of Roger Michael Needham, page 45, lines 4-9:

> Q. Do you believe this RFPJ [sic] J-1 is necessary to protect anything other than keys and the locations of keys?
>
> A. I don’t believe it is. It is to protect keys and the locations of keys from being indirectly inferred.

(Needham Dep. Tr., March 1, 2002 at 45).

to allow it not to disclose APIs relating to its Windows Media Player – even though Microsoft's Remedy defines Windows Media Player to be Microsoft Middleware (*see* RPFJ Sections VI.J and VI.K) for which API disclosure obligations would otherwise apply (*see* RPFJ Section III.D.). In fact, it appears that Microsoft has already done so on at least one occasion. David Richards of RealNetworks testified that Microsoft has denied his company's requested access to the Windows Secure Audio Path ("SAP") technology that is used to protect media content from piracy. Microsoft has made the SAP available only for purposes of interoperating with Windows, but has not generally licensed it to competing middleware vendors such as RealNetworks. *See* David Richards Direct Testimony at ¶¶ 75-78.

127. While the use of DRM is currently focused on media, Microsoft may in the future require DRM to interface with all components of its operating system. Since software components, applications, and middleware are all copyrighted "content," digital rights management might apply to any software component or to the use of any API. Section III.J.1 does not constrain Microsoft from broadly implementing DRM technology for the underlying components of its operating system and would allow Microsoft effectively to keep a great deal of its APIs, documentation, and communications protocols undisclosed. Further, by implementing DRM on a wide scale, even if Microsoft disclosed the APIs, a key would still be needed to use the APIs effectively, and the provision does not require Microsoft to give out the keys necessary to interoperate with the operating system. This exception could prevent any use of the operating system APIs by non-Microsoft software for anything relating to copyrighted material.

128. The States' Remedy does not contain a broad security exception like that found in Section III.J.1 of Microsoft's Remedy. Rather, it requires Microsoft to disclose the information necessary for interoperability while allowing Microsoft to ensure the secure transmission of information. Nothing in the States' Remedy would require Microsoft to divulge information that would result in the disclosure of "keys or the locations of keys" that hackers could not find by other means. At the same time, nothing in the States' Remedy would exempt Microsoft from disclosing interfaces that are needed for interoperation based on technically unjustified "security" issues. In my opinion, the disclosure of information required by the States' Remedy would not interfere with the security of a properly designed computer software or system.

IX. THE COMPONENTIZATION OF MICROSOFT SOFTWARE CODE SUPPORTS THE FEASIBILITY OF THE STATES' REMEDY UNDER SECTION 1

129. As a result of my training and experience in academia and software development, I have developed an understanding of commonly used and effective methods of software development. One of the most basic concepts in software development is that well-developed software is written in modular fashion. This is how I develop software, and this is how I teach and train my undergraduate and graduate students to write software code and programs. I have published papers ("A Critique of Standard ML," "Hierarchical Modularity," "Separate Compilation for Standard ML," and others) on the theory and practice of modular software development.

130. Modular programming entails the concept that related functions should be contained within the same unit of programming code and that unrelated functions should be developed as separate units of code. This software componentization is necessary so that

the code can be properly maintained and used by different programs. In the 1960's, before the widespread use of modular programming, experience showed that any line of source code might possibly interact with any other line, so that in a million-line program there would be perhaps a trillion potential interactions. With modular programming, on the other hand, each line of code interacts only with other code in the same module, and with code in APIs connected to that module. The field of software engineering has employed modular programming as an accepted programming standard for over twenty years.

131. An important principle of modular software development is the interchangeability of implementations: a module that implements (matches) a particular API can be replaced by a different implementation that's functionally equivalent.

132. One advantage of modular development is that it is generally possible to determine which of the APIs or modules are needed by a particular piece of software. There are dependence-analysis tools and software-structure analysis tools that help to automate this determination. This means that as a technical matter, the modules serving to support Microsoft's middleware should be removable without causing disruption to the functionality of the remaining operating system.

133. I am of the opinion that the code underlying Microsoft's software platform products is most likely written in modular fashion. I am of this opinion for several reasons. First, I am aware that at least some aspects of the Microsoft's code are modular. For example, in prior proceedings before the District Court in this case, Professor Edward Felten indicated that he was able to locate an interface between the Windows operating system

and Microsoft's Internet Explorer browser. The presence of an interface indicates that the browser was developed as a separate module. Professor Felten also showed that for almost all purposes, the Internet Explorer module could be replaced by the Netscape Navigator module, because they both matched the same API.

134. Second, because of the extremely large and complicated nature of Microsoft's operating system code, it is my opinion that it must be written according the accepted standards of modular design in order to allow for the proper maintenance and implementation of the code. A software program of the size and complexity of the Windows operating system product could not effectively be maintained if it were not modularized to a large degree.

135. Third, Microsoft ships its operating system and applications partly in the form of "DLLs," that is, dynamically linked libraries, which are software modules in the sense that I have been using that term. This is further evidence that the software is modular.

136. Fourth, Microsoft markets a componentized version of its Windows XP operating system product, called Windows XP Embedded, that consists of the same binary code as Windows XP. Windows XP Embedded allows OEMs to select components from Windows XP to install on the OEMs' hardware. Among the components that OEMs may select for inclusion are device drivers, certain Microsoft Middleware Products, and other software. Microsoft provides a tool called the XP Embedded Target Designer that performs dependency analysis, showing the OEM how including one component will require other components to be included.

137. Exhibits G, H, and I are screen shots from the Target Designer that I have printed. Exhibit G explains the benefits of componentization as it applies to the Windows XP

operating system. Exhibit H explains that "Windows XP Embedded contains exactly the same binary files as Windows XP Professional. Therefore, it supports all of the same features." This means that the Windows XP Embedded operating system is built from exactly the same computer source code as the Windows XP Professional operating system. Windows XP Professional is the latest commercially available version of Microsoft's Windows Operating System. Microsoft touts that XP Embedded offers a "100% 'Componentized' version of OS" that "uses the same binaries as retail OS release."[16] Because the binaries (the actual executable code loaded onto a personal computer) are the same, Windows XP Embedded supports the same features as Windows XP.[17] Exhibit I shows that Windows Media Player and Internet Explorer are removable components of Windows XP Embedded.

138. Section 1 of the States' Remedy requires Microsoft to create an "unbound" version of the Windows operating system product, i.e., one from which certain Microsoft Middleware Products can be removed and replaced with competing middleware products. (Of course, Microsoft can also continue to offer its current Windows Operating System Product that has its middleware bound to the operating system code.) Microsoft's technical ability to create XP Embedded (and the Target Designer) is another piece of evidence not only that Section 1 of the States' Remedy is technically feasible, but that Microsoft has already done much of the engineering work necessary to comply with this provision.

[16] *See* Windows XP Embedded, Technology Briefing, March 2001, at 5, attached as Exhibit J.

[17] "Note that we support 100% of Windows XP – lack of coverage [in this document] doesn't mean its not in the embedded product." *See* Exhibit J at 12.

X. MICROSOFT'S REMEDY PROVIDES MICROSOFT AN UNNECESSARY TECHNICAL EXCLUSION

139. Section III.H.2.b provides a technical exclusion that allows Microsoft to override a user's choice of a non-Microsoft Middleware Product – *i.e.*, to replace it with a Microsoft Middleware Product – when the non-Microsoft Middleware Product "fails to implement a reasonable technical requirement (*e.g.*, a requirement to be able to host a particular ActiveX Control)." This exception is technically unjustified. And it provides Microsoft with unfettered discretion to overrule a user's decision to use a non-Microsoft Middleware Product because Microsoft reserves to itself the ability to define and control the technical requirements that must be met.

140. For example, Microsoft could easily specify thousands of "technical requirements" that must be implemented by non-Microsoft Middleware Products. Any particular middleware (Microsoft or non-Microsoft) may have a technical need to implement only a few of these technical requirements. Many applications will run perfectly well on such middleware that, for example, hosts only a subset of ActiveX Controls. A third party who packages a combination of Microsoft and non-Microsoft components may only need to meet a small subset of the thousands of technical requirements in order for the entire package to function properly. But the fact that the non-Microsoft Middleware Product does not support all Microsoft's technical requirements, many of which are unnecessary, would provide Microsoft with the ability to override the user's choice of the non-Microsoft Product in favor of its own Microsoft Middleware Product. This is particularly problematic because the list of technical requirements is determined at Microsoft's discretion.

141. I have just explained a scenario in which a package of middleware and application components functions perfectly well even though some of the "technical requirements" established by Microsoft are not satisfied. But even in a case where applications do require technical features not supplied by a particular middleware, Section III.H.2.b would be unnecessary.

142. It is easy for software to give a diagnostic message to the user when a middleware does not provide a particular functionality. Microsoft itself already implements such diagnostic messages: when I run Internet Explorer to browse web pages, I do so with ActiveX disabled (to protect myself from viruses), and occasionally I get the message, "Your current security settings prohibit running ActiveX controls on this page. As a result, the page may not display correctly." I may also get the message, "An error occurred inside a plug-in contained on this page." That is, the problem is identified, explained to the user, and prevented from causing harm. Microsoft has given users an ability (in Explorer's Internet Security Preferences) to decide which ActiveX controls should be enabled; here, Microsoft recognizes that users want choice over which technical requirements should be satisfied in their local environments. This is very different from the Section III.H.2.b, which permits Microsoft to disconnect an entire (non-Microsoft) middleware because it doesn't support a single ActiveX control.

XI. MICROSOFT'S REMEDY FAILS TO REQUIRE MICROSOFT'S APPROPRIATE ADHERENCE TO INDUSTRY STANDARDS

143. Even if Microsoft's Remedy required Microsoft to make all necessary disclosures, it does not address the problems created by the manipulation and pollution of industry and de facto standards that are relied upon by third parties for interoperability. If Microsoft

documents its APIs, protocols, and other technical information, either through disclosure or submission to industry standards bodies, it encourages others within the industry to rely on this information as the means of achieving interoperability with Microsoft products. But Microsoft can, and has, subverted this reliance by not abiding to these standards, causing interoperability difficulties as significant as those created through nondisclosure. In addition to addressing Microsoft's nondisclosure of necessary information, an appropriate remedy must therefore prevent the abuse of standards in order to provide meaningful relief.

144. Standards establish uniformity with regard to a particular area of the computer industry, such as communications protocols, authentication, or file formats. These standards allow parties to develop products that will properly interoperate with others conforming to the same standard. One method through which standards may be created is the formal recognition and publication of standard specifications by a standards body. For example, the Institute of Electrical and Electronics Engineers ("IEEE"), publishes wide-ranging industry standards relating to electrical and information technologies. Standards may also originate when the specifications created and used by a single company are disclosed and then become widely adopted by the industry as a whole. These standards are commonly referred to as de facto standards.

145. Based on my experience studying and developing software, standards represent one of the important methods for software developers to achieve cross-platform interoperability. By providing proprietary and often undocumented extensions to standards, Microsoft can mislead third parties to believe that the publicly documented specifications of a standard will ensure interoperability when in fact interoperability cannot be achieved without the

extensions added by Microsoft. This behavior can cause developers to spend great resources under the mistaken belief that their software will achieve the benefits of cross-platform interoperability with those sharing the same standard, only to ultimately achieve (at best) limited interoperability.

146. Disclosure requirements address a key problem of interoperability, but additional requirements on standards are necessary to prevent the corruption of this disclosed information by preventing any confusion that it represents the fulfillment of standard specifications when in actuality it does not.

147. Section 16 of States' Remedy addresses this concern by ensuring that Microsoft adheres to the standards that it purportedly supports. Section 16.a requires that if Microsoft publicly announces that it adheres to a standard, it must in fact do so. (Of course, Microsoft would not be required to support any particular standard.) Section 16.a permits Microsoft to extend a standard (for which they have claimed compliance) so long as it also continues to support the non-extended standard. However, in that circumstance, Microsoft cannot require third parties to use its extended implementation of the standard, and Microsoft's own Platform Software must still be able to interoperate with the non-extended implementations of that standard.

148. In reading Section 16.a, it is helpful to understand the difference between a standard and the implementation of a standard. The difference is analogous to the difference between an API and the implementation of a component that supports the API, as in Exhibit E where "testprime.h" is the API and "testprime.c" and "divides.c" are the implementation. If the standard is a communications protocol, for example, the standard specifies what

kinds of messages are exchanged, and the implementation is software that actually sends and receives the message.

149. Provision 16.b provides some flexibility to Microsoft in the case that a standard as used in industrial practice differs from the formal definition of that standard.

150. It is my opinion that provision 16 is necessary to allow third parties to develop software that effectively interoperates with portions of Microsoft platform software that utilize industry or de facto standards, and it will help ISVs to make informed design decisions about the portability of their software.

I declare under penalty of perjury of the laws of the United States that the foregoing is true and correct.

Executed this ___ day of March 2002.

____

Andrew W. Appel