FENWICK & WEST LLP

Baystate Holding—Technical Interfaces Not Copyrightable

By Mitchell Zimmerman (April 1997)

A district court in Boston has rendered an important and thoroughly reasoned decision on the copyrightability of computer software technical interfaces, holding that critical elements of such interfaces cannot be protected by copyright law. The case provides significant guidance on how much control the creators of computer hardware, operating systems, and language products can assert over the development of compatible or competitive software by barring others from employing the technical interfaces contained within their original works.

In *Baystate Technologies, Inc. v. Bentley Systems, Inc.*, 946 F. Supp. 1079 (D. Mass. 1996), decided December 6, 1996, the district court of Massachusetts held that the data structure names and the organization of data structures of a mechanical computer aided design program are not copyrightable. The program elements at stake in *Baystate* are in important respects indistinguishable from the larger class comprised of names that must be used and parameters, formats and structures that must be employed in order to invoke the functionality of particular proprietary microprocessors, operating systems, computer languages and application programs. The *Baystate* decision therefore supports the broader conclusion that "using" or "copying" such labels and structures in order to create compatible works is lawful under the Copyright Act.

Most "software interface" decisions in recent years have addressed the scope of protection given to user interfaces and their command hierarchies.[1] And the relatively few cases with strong implications for technical interfaces have provided only limited guidance because the opinions were factually unclear or did not involve technical interfaces, cryptically reasoned at best, or inconclusive in their ultimate holdings.

Baystate is important for two reasons. First, the protectability of technical interface elements required to create compatible works was clearly posed and resolved in the case. Second, the *Baystate* opinion is more completely (if still imperfectly) reasoned than any of the other decisions that have treated technical interface compatibility issues. Since the parties recently settled the case on appeal, the district court opinion stands as the decision most sharply addressing the protectability of technical interfaces. Before we examine the decision in *Baystate* more closely, let us first place the case in its legal setting.

Unresolved Issue: "Copying" of Functional Elements

The backdrop for this battle in the war over "compatibility" is familiar territory: once Congress made clear that computer programs could be protected under copyrightable law, an unresolved and perhaps irresolvable tension in the law arose. On the one hand, copyright law generally does not protect functional aspects of works, and procedures, processes, systems or methods of operation have been dubbed unprotectable by statute.[2] On the other hand, computer programs are inherently functional works, and the most important and creative aspects of computer programs can often be appropriately described as procedures, processes, systems, or methods of operation. Given the job of squaring the circle, the courts have adapted to the task such doctrinal tools as the idea-expression continuum, merger, and *scènes à faire* (as well as refining infringement standards and fair use and other defenses), and have slowly wrought a body of law defining the scope of rights in various elements of machine-readable works.

But, however adequate that process has been in other contexts, it has not yielded a clear body of decisions in one area of critical importance: the protectability of technical interfaces or specifications. Late-comers to particular software markets consider themselves obliged by competitive and/or technical requirements to create new works that "build on," "are interoperable with" or "copy" (pick your epithet) various "technical" aspects of earlier works, thereby raising the issue of the protectability of the "copied" elements. The cases have offered little guidance on the extent to which later works may be made consistent or compatible with earlier programs or devices by using (or "copying") the function names, parameters or structures needed to invoke functionality, by using the input formats needed to make data files created with an earlier program accessible, or by employing the interface standards that must be adhered to if a new product is to run on a proprietary platform.

What is a "Technical Interface"?

The "technical interfaces or specifications" which we are treating as a single category include a range of program features or elements. Some examples:

Operating System Calls. To make a "system call" (invoke certain

1 *See, e.g., Lotus Development Corp. v. Borland International, Inc.*, 49 F.3d 807 (1st Cir. 1995), *aff'd without opinion by equally divided Supreme Court*, 116 S. Ct. 804 (1996); *Ashton-Tate v. Ross*, 916 F.2d 516 (9th Cir. 1990); *Brown Bag Software v. Symantec Corp.*, 960 F.2d 1465 (9th Cir.), *cert. denied*, 113 S. Ct. 198 (1992); *Autoskill Inc. v. National Educational Support Sys.*, 994 F.2d 1476 (10th Cir.), *cert. denied*, 114 S. Ct. 307 (1993).

2 *See* 17 U.S.C. § 102(b); *see, e.g., Apple Computer, Inc. v. Microsoft Corp.*, 799 F. Supp. 1006, 1023 (N.D. Cal. 1992) ("[p]urely functional items or an arrangement of them for functional purposes are wholly beyond the realm of copyright"), affirmed 35 F.3d 1435 (9th Cir. 1994).

functionalities of a particular operating system), the source code of an application program must use the same "command' terms (roughly, "words" such as "AfxGetResourceHandle" or "SetWindowPos") as are recognized by that operating system. Is the collection of such commands, each commonly a coined semi-mnemonic term, protectable either by itself or in conjunction with other system elements?

Parameter Structures. In order for a software engineer who is developing an application program to employ the functionality of a computer language such as "C" or an operating system, her program must issue instructions to the system in accordance with the system's "rules" or syntax. So, for example, to draw a line on a screen, it is necessary to state in a specified order, using specified terms, within a specified range, the starting and ending points, color, thickness and other characteristics of the line. Are such parameter structures or rules copyrightable subject matter?

Input and output formats of application programs. Application programs require that data be input to them (for example, mechanical data for a structural analysis program), and often produce data as an output for use by other programs (for example, data for a spreadsheet). Input and output formats specify the ordering, structure and type of data that must be input or that can be output by the program. Input data may, for example, be input manually, passed in from another program, or stored in a file. When stored in a file, input or output data is stored in a specified "file structure," and for data to be passed from another program, usually either that program must employ the same file employing the same formats or structure or must be translated into a readable structure. Are such file structures and formats protectable?

Although, in these examples, the various elements are associated with different kinds of programs—operating systems or other system software, languages, application programs—in reality, the various kinds of elements (or comparable features) cross those lines: operating systems as well as application programs may dictate file structures; parameter structures or their equivalents may be found in computer languages as well as in applications and systems software. What is common to these features on a "functional" level is that they involve the "machine—machine" interface, the place where the software or firmware for two programs or devices exchange information with each other, and not the "human—machine" interface, the place where people interact with a machine running a program. What is common to these features on a more "descriptive level" is that these technical interface elements or features are commonly comprised of sets of labels (command terms) or "words" that must be used and/or of rules that must be observed by those who create programs to work with other programs or machines. When we speak of "technical interfaces" in this article, then, we are speaking of the features or elements necessary for machine—machine or program—program relationships.

What is also common to technical interfaces is that copying or employing them may be necessary in order to create new works that are "compatible" with the original works in some sense or other. Many cases have, of course, stated that the "need for compatibility" may render an element unprotectable or require that its use be held noninfringing.[3] But "compatibility" has had many meanings in software copyright law, and courts have variously accepted or rejected "compatibility" as a basis for copying, depending in part on what was meant by the term and on the context. Before considering *Baystate*'s contribution to clarifying this situation, it will be instructive to consider the principal cases in which the protectability of technical interfaces has been presented hitherto. Six cases have ruled on comparable issues and, predictably, were split in their outcomes.

Cases Indicating Technical Interfaces May Not Be Protectable. Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1992). This case out of the Ninth Circuit focused on an important "fair use" issue, whether Accolade's intermediate copying through disassembly of Sega object code in order to discover the functional requirements for compatibility with the Genesis game console represented "fair use." Defendant Accolade sought to understand those "functional requirements" so that it could prepare unlicensed games that would run on Sega's platform, and indeed its games did embody such functional specifications. Interestingly, plaintiff Sega did not charge Accolade with infringement for copying or employing Sega's functional specifications. The issue was therefore not squarely framed by the complaint, and was not directly before the Ninth Circuit. Nonetheless, the Ninth Circuit addressed the issue and resolved it.

The appellate court held that such copying and reverse engineering, or disassembly for discerning "system interface procedures," is a fair use when it is "the only means of gaining access to [the] unprotected aspects of the program," namely ideas and functional concepts required for compatibility. (977 F.2d at 1520.) In reaching that ruling, the Court necessarily and expressly held that the functional requirements for compatibility with [Sega's] Genesis console [are] aspects of Sega's programs that are not protected by copyright. 17 U.S.C. § 102(b)."[4] (977 F.2d at 1522.) In so holding, the Ninth Circuit evidently resolved the issue under discussion in this article.

3 *Gates Rubber Co. v. Bando Chem. Indus., Ltd.*, 9 F.3d 823, 838 (10th Cir. 1993); *Bateman v. Mnemonics, Inc.*, 79 F.3d 1532, 1546-47 (11th Cir. 1996).

4 The cited section of the Copyright Act is, of course, the one which provides: "In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."

The Ninth Circuit appears to have resolved the issue, but its treatment was so cryptic—involving no more discussion than that quoted above—that the decision stands for little more than the result, and even that is not perfectly clear. First, although the court mentions "the functional requirements for compatibility with the Sega console," namely, "system interface procedures," it is not entirely clear from the appellate decision precisely what the court was talking about. Nor does the district court opinion illuminate the issue any further. Second, although nothing in the opinion suggests that the court was merely assuming the unprotectability of the "system interface procedures" because it was uncontested in that case, neither does the court indicate that the protectability of such interface procedures was indeed at issue. In the end, we believe there is little doubt that the ruling on fair use contains a holding that the system interface procedures in the case were unprotectable. But the failure to be more specific or to discuss the issue has limited the precedential force of this ruling.

Bateman v. Mnemonics, Inc., 79 F.3d 1532 (11th Cir. 1996). In this case, the defendant developed an operating system competitive with the copyright holder's operating system. In order to run application programs written for the original system, defendant's "interface specifications" or technical interface had to be more or less the same. The Eleventh Circuit Court of Appeals, consequently, had to consider whether the interface between an operating system and applications written to run under that operating system was protectable under copyright. [5]

The court did not issue a definitive ruling on the ultimate issue, and remanded the case for further proceedings, but its discussion was illuminating, elusive and in the end evasive. On the one hand, the Court of Appeals opined that "interface specifications" are not uncopyrightable "as a matter of law." This determination suggests that the Eleventh Circuit was rejecting the argument that interface specifications (at least of the kind before it) are simply uncopyrightable subject matter. On the other hand, the decision states that copying for purposes of "compatibility" may negate copyright liability, because compatibility-required elements are not copyrightable and also, perhaps, because copying such elements constitutes fair use. (79 F.3d at 1546-47 & fns. 28, 30, 33.)

What kind of compatibility was the Eleventh Circuit talking about when it referred to "compatibility requirements"? The term has a range of potentially relevant meanings, which may for convenience be herded into two main groupings:

User expectation or market compatibility: This refers to "requirements" for similarity arising out of the expectations and preferences of end-users that new products will work the same way as the old products they are used to (*e.g.*, users may expect that a tape player's play, stop and fast forward and reverse buttons will be in a certain order).

Functional or technical compatibility: This refers to "requirements" for similarity arising out of technical mandates, if different products are to work with each other (*e.g.*, an extension cord had better have at least as many holes as the plug that goes in it has prongs).

These groupings are rough; there are "compatibility" considerations that straddle the two categories; and each group includes a variety of animals that are significantly different from each other. Nonetheless, it is important to distinguish between consumer expectations or demands (such as preferences for a certain look and feel in the final product) and technical demands (specifications that must be met if one program is to work with another at all, or if one program is to work with applications or data files that were created under or by an earlier program).

Although the Court of Appeals chided the district court for "fail[ing] to define the important term 'compatibility'" for the jury (*id.* at 1546), the Eleventh Circuit also omitted to offer a direct definition. Nonetheless, it is quite clear that when the Eleventh Circuit found that "external considerations such as compatibility may negate a finding of infringement" (*id.* at 1548), the court was referring only to functional compatibility. *See, e.g.*, discussion at 1546-47; footnotes 28 (referring to "[c]ompatibility and other functionality challenges") and 30 ("'Compatibility becomes an issue when a firm wishes to create a single new component designed to operate with elements of a preexisting system"). Since the Eleventh Circuit returned the case to the district court for further consideration because the instructions given the jury in the first trial had not made it sufficiently clear that "compatibility" can negate liability, the Eleventh Circuit indicated that the need for such technical compatibility "may" negate liability in such a situation. But given the absence of any guidelines for when such compatibility needs may or may not result in unprotectability, it is difficult to know what to make of the decision.

Mitel Inc. v. Iqtel, Inc., 896 F. Supp. 1050 (D. Col. 1995). This case did not directly involve computers or computer programs per se, and hence there can be some argument about whether it applies to software technical interfaces at all; also, it might be argued that *Mitel*

5 Although the court used the term "interface specifications," the court did not say just what it meant by that term. Presumably, in view of the compatibility issues at stake in the case, the court was using the term to refer to the technical interface—and most likely the system calls— to the operating system, which the allegedly infringing operating system would have had to copy in whole or in part to achieve compatibility with the application programs.

involves a user interface and user compatibility expectations, rather than a "technical interface" and technical compatibility requirements. Notwithstanding the imperfect fit, the case supports an argument that technical interfaces are not protectable.

In *Mitel*, a federal district court in Colorado held that command codes used to program a manufacturer's telephone call controller were not copyrightable. These "command codes" were three and four digit numbers or letters that engaged a particular function (*e.g.*, automatic redial or speed dialing) of the call controller. The codes were not directly accessed by the telephone companies that bought the devices (let alone telephone service customers), but were used by technicians who programmed the controllers in the process of installing them. Once the devices were coded and installed, telephone service customers were able to access the functionalities which the call controllers enabled, though the telephone customers did not themselves enter the call controller codes). Plaintiff Mitel conceded that the command codes were not a computer program.

Defendant did not dispute that it had copied plaintiff's three and four digit command codes for use in its telephone call controller. Defendant maintained that for its call controller to be competitive, it had to be compatible with plaintiff's call controller since plaintiff had 75% to 90% of the call controller market, and technicians expressed reluctance to learn new command codes. Plaintiff argued that its codes were protected because they represented a "creatively chosen set of alpha numeric codes." (*Id*. at 1054.) Defendant responded that the command codes were an unprotectable method of operation under 17 U.S.C. § 102(b), and that the *scènes à faire* and "fair use" doctrines also applied. The court agreed with the defendant on all counts, holding that the command codes were not subject to copyright protection because they were "a procedure, process, system, and method of operation." (896 F. Supp. at 1055.)

Because the initial "users" of the command codes at issue were the technician installers, it may be tempting to liken the codes to user interfaces elements. However, once programmed, the call controller codes are continually and automatically invoked by the phone companies' systems, and the chosen codes are the only ones recognized by the call controllers as invoking the selected functionality. The codes can therefore reasonably be compared to such technical interface elements as the header names of an applications programmer's interface, in that both are comprised of sets of short "terms" that technical personnel must program into a device or program in order to invoke an underlying functionality. So viewed, the telephone technician/installers play the same role as software programmers, and should not be seen as end-users.

Most of the *Mitel* court's reasoning, if the analogy holds, would apply directly to the arguments that may be made in a technical interface infringement case. First, the court rejected the argument that the codes were "original" in the copyright sense or represented protectable expression merely because there were alternative "expressions" that were available for the idea of function-invoking codes, and that the creator of the original work therefore made "choices." (896 F. Supp. at 1055.) Second, the court emphasized the fact that the codes were a means of invoking functionality: "The command codes in this case simply act as a key to unlock the inner functions of the call controller." (*Id*.) Third, the court interpreted and relied on the *scènes à faire* doctrine. Because *Mitel* was the dominant player in the industry, its command codes had become a "common practice in the industry." Iqtel therefore needed to copy the command codes to ensure uniformity and for reasons of efficiency; hence, the codes were not protectable under the *scènes à faire* doctrine. Finally, the district court held that Iqtel's use of the codes constituted fair use, on the theory that accommodating the demands of the technicians and bringing a new competing product to the market represented a "'legitimate non-exploitative purpose' for copying the command codes." (*Id*. at 1057.)

Cases Indicating Technical Interfaces May Be Protectable.
Control Data Systems, Inc. v. Infoware, Inc., 903 F. Supp. 1316 (D. Minn. 1995). On a motion for a preliminary injunction, the district court held that defendant's "emulator" program likely infringed copyrights in the technical interface and source code of Control Data's network operating system (NOS). Defendant's program was designed to permit applications software that was originally written for Control Data's Cyber computers and operating system to be used on other manufacturers' hardware. The alleged similarities between the NOS operating system and defendant's emulator software included (1) 2,000 lines of copied NOS source code, (2) the NOS input and output formats, (3) NOS file layouts, (4) NOS source code parameters and (5) NOS commands.

The opinion sets forth only a limited exposition of the facts because much of the evidence was under a protective order, and the basis of the decision is not entirely clear. Nonetheless, it appears that the district court relied on the source code copying (see 903 F. Supp. at 1322-24), and the court did not discuss or analyze whether the other elements were protectable. The "protectability" discussion in *Control Data* focused on a single point: whether Infoware's "need" to create a "NOS-compatible" operating system represented an "externality" which, under the merger doctrine, rendered the copied elements unprotectable.

The court's disposition of the issue illustrates the important point that there are two perspectives from which one can address "compatibility"—that of the original developer or that of the latecomer—and that this choice of perspective commonly determines the result.[6] The court held that the "compatibility" issue should be considered from the perspective of the original developer. The elements at issue were not dictated by the "idea" of making an operating system that would function like NOS and the need for "NOS-compatibility" was not an "externality," the court concluded, because the relevant idea was rather that of making a system compatible with Control Data's Cyber computer (the problem faced by the original developers). Since Control Data's NOS program did not represent the only way to make a Cyber-compatible operating system, NOS-compatible elements were not filtered out of the analysis. By shifting the time perspective considered relevant, the court effectively negated the need for compatibility based on end-user and industry expectations without (at least nominally) rejecting the view that "compatibility requirements" are a constraint. In the terms discussed earlier, another way to put this would be that the court seemingly recognized functional compatibility as a possible basis for rejecting liability, but not user expectation or market compatibility.

Be that as it may, it does not seem that the court focused on whether the "copying" of elements other than source code implicated any copyrightability issues apart from compatibility. The omission may have been reasonable in light of the limited purpose of the proceedings, whether to grant a preliminary injunction, and the evident sufficiency of plaintiff's showing of source code copying to sustain an injunction. But the sufficiency of the source code evidence means that the case contains no clear holding or exegesis and provides no real guidance on whether operating system parameters, commands, input and output formats and file layouts are copyrightable subject matter.

CMAX/Cleveland, Inc. v. UCR, Inc., 804 F. Supp. 337 (M.D. Georgia 1992). Plaintiff CMAX was the author of a computer program called "RMAX," which was used to input, store, process and retrieve information incident to the "rent-to-own" furniture and appliance business. Defendant UCR originally licensed CMAX's software, and set out to develop an in-house program that would perform the same functions as RMAX, work like RMAX, and be compatible with files and records previously created using RMAX, in order to avoid CMAX's license fees. Defendant UCR developed a program of identical design, essentially a clone program, not for sale to third parties nor because of the market demands of other end-users, but for its own use. In order to maintain compatibility with its existing RMAX data files, defendant studied the file structure and file names of the RMAX program in detail and replicated them in its own program. In addition, defendant replicated many of RMAX's screens and reports as well.

The court ruled that the file layouts or structures (including the field definitions contained therein), record layouts, file names, naming conventions, transaction codes, screens and reports of the plaintiff's program constituted protectable expression because they were dictated neither by industry standards, by efficiency, nor by the need for the program to interact with the central host computer it was designed to operate with. (*Id*. at 354-55.) Though the opinion is not 100% clear about whether certain of the items at issue are in the user interface or technical interface category, at least some of the copied elements which the court held protectable were technical interface elements.

"File structures": What the court referred to as "file structures" were comprised of file names, sequences of fields, and field names or field definitions. (Pp. 348-49, 354-55.) The court found that these "structures" for inputting data were not dictated by market forces and that it was not functionally required that they follow a particular order. (Pp. 354-55.) The court held that because such file structures collect and organize information entered by the user, they are not merely "blank forms," and that there is protectable expression in the selection and arrangement of the field definitions. The field names and sequences function as technical interface items. (P. 354.)

"Screens and reports": These kinds of user interface elements have often been held protectable, and here apparently involved the same input formats and structures as discussed as file structures. Although they are user interface elements in that the user sees and works with these elements when entering data and reviewing results, as part of the stored files these elements become a technical interface insofar as they are used and can be read by other programs. Since (the court held) the field definition sequence in the screens and reports was not dictated by the industry and since the data fields could have been arranged in any number of ways, they represent protected expression. (804 F.Supp at 354.)

"Transaction codes": As defined by the court, these sound like technical interface elements— "A transaction code is a randomly selected, alphanumeric sequence of characters that indicates to the computer what steps should be executed in a given situation or 'transaction' when the code is transmitted" (*id*. at 349 n.8)—but it is apparently the end-user of the program who directly employs the transaction codes (p. 355). Again, like the *Mitel* telephone system

6 For a more thorough discussion of these two perspectives in recent cases, see David L. Hayes' monograph, *A Comprehensive Current Analysis of Software "Look and Feel" Protection* (Fenwick & West LLP, 1997), pp. 129-30. The author is indebted to Mr. Hayes for his helpful feedback on an earlier draft of this article.

command codes, transaction codes are somewhat analogous to header names or certain other technical interface elements, so the *CMAX* holdings on these elements are not irrelevant. Rejecting the argument that the defendant's employees' training in the use of the RMAX transaction codes was an externality that would limit protection of the codes (*i.e.*, rejecting what we have termed "user expectation compatibility" as a defense), the court found that the transaction codes were a major element of the program's design, that they were not dictated by efficiency or industry demands, and hence that they were protected by the program's copyrights.

Engineering Dynamics, Inc. v. Structural Software, Inc., 26 F.3d 1335 (5th Cir. 1994) ("*EDI*"). Like *CMAX,EDI* involves the input formats employed by end-users. In *EDI*, the Fifth Circuit held that the input formats of an application program designed to solve structural engineering problems constituted copyrightable subject matter. Engineering Dynamics defined a specific set of input formats for use with its program, in which the user could enter the required data, including construction details and anticipated environmental and other external forces. Structural Software copied many of Engineering Dynamics' input formats. The court ruled that the input formats constituted copyrightable subject matter and that Structural Software infringed Engineering Dynamics' copyright.

The case may be informative primarily by way of analogy. The court described the formats as "quasi-textual," consisting of a "series of words and a framework of instructions that act as prompts to the user to provide relevant data" to the program so it can perform a series of sophisticated structural analyses. (*Id*. at 1342.) The court stated that "generally functional interfaces that directly teach or guide the user's independent decisions are more expressive than functional interfaces that lack these qualities." (*Id*.) Based on its finding that [plaintiff's] interface "imparts knowledge" by telling the user which data to collect as well as the order of collection, the court concluded that plaintiff's input formats showed enough original expression to warrant protection. (*Id*.at 1346.) Although, again, the input formats ruled protectable were part of the user interface (and, indeed, it was the copyright in the user manual which the court held infringed (*id*. at 1339)), the input formats (when stored) become embodied in file formats which are part of the technical interface. The interface ruled protectable in *EDI* therefore partakes of both user interfaces and technical interfaces. Since the holding that the formats are protectable relies on the "user interface" aspect of the formats (that they impart knowledge to the user), although the court *de facto* also protects the formats as embodied in the technical interface of the program, the court does not discuss or rule on the protectability of the technical interface in isolation from the user aspects.

Summing up the body of most nearly relevant cases on the protectability of technical interfaces, we find one case clearly (if tersely and without analysis) holding such interfaces to be unprotectable (*Sega*); two cases that presented the issue but failed to resolve it (*Bateman* and *Control Data*); one case that involved a technically analogous issue and resolved it against protectability (*Mitel*); and two cases that involved mixed or cross-over user/technical interface elements (input formats), and held them protectable (*CMAX* and *EDI*). Enter *Baystate Technologies*.

Factual Background of the *Baystate* Dispute

Plaintiff Baystate owned the copyrights for a mechanical computer aided design (CAD) program. Such programs "enable architects, engineers and other design professionals to design and alter the design of buildings, mechanical devices and electronic equipment using computers as drafting devices. They can then produce blue prints and other design drawings through their computers." (*Baystate*, 946 F.Supp. at 1082.)

Defendant Bentley Systems, Inc., a competitor in the market for such programs, sought to create a translator program which was to employ certain elements from *Baystate*'s program. The district court identified the copied elements as the "names of the so-called data structures and the organization of the files within the data structures," including, "more specifically, the words and abbreviations used to describe the files contained within the data structure and the data structures themselves." (*Id*. at 1087, 1088.) These structures were found in the header files of plaintiff's program.

District Judge Nathaniel M. Gorton's opinion is not entirely clear about why Bentley was creating a translator. But he does observe that "data translators are common in the CAD market because users of CAD products commonly employ more than one CAD system to perform their necessary tasks and, as a result, often transfer information between various CAD systems. This need has created a demand for translators of all kinds in the CAD market . . .." (*Id*. at 1082.)

The opinion is silent on what the different kinds of translators are in the CAD market, and silent on a related question which is suggested by the opinion: Why is there is a lawsuit at all if Bentley is just one more company preparing a CAD translator in an industry seemingly rife with them? We can easily infer the answers to these questions. There is clearly a range of complementary CAD-related products, and translators are doubtless common in the CAD industry for purposes of interoperability. That is, translators are needed in order to allow other products that do something different from *Baystate*'s CADKEY (for example) to accept CADKEY's data output and work with it. It may be

in the interest of authors like *Baystate* to acquiesce in the creation of such complementary products, because they make the original tool more useful, and their customers are more likely to remain committed to the original tool and to new versions of that tool if they become committed to working within a web of related, compatible products.[7] Bentley's product, however, was apparently not intended to go with CADKEY, but to compete with and replace it. (*Id.*) And Bentley's translator was presumably intended to facilitate the migration of *Baystate* customers from Baystate's CADKEY program to Bentley's Microstation program by allowing data files created under CADKEY to be read by and written to Bentley's program.

Copyright Infringement Analysis in the First Circuit

To prove a claim of copyright infringement, a plaintiff must prove ownership of a valid copyright and that the defendant "copied" the protected work. In order to prove "actionable copying," the First Circuit employs a two-step analysis. *Lotus Development Corp. v. Borland International, Inc.*, 49 F.3d 807 (1st Cir. 1995), *aff'd*, 116 S. Ct. 804 (1996). First, the copyright holder must prove factual copying. Factual copying can be shown either by direct evidence or circumstantially, by proving that the defendant had access to the copyrighted work and that there is "a probative similarity" between the two works. (49 F.3d at 813.) If factual copying is shown, the copyright holder "must then prove that the copying of copyrighted material was so extensive that it rendered the offending and copyrighted works substantially similar." (*Id.*)

The District Court's Analysis

In sections of the opinion of only passing interest, the court found that *Baystate* owned valid copyrights to CADKEY; that CADKEY was, taken as a whole, protected by those copyrights; that the entity which prepared defendant Bentley's translator had access to the relevant files of CADKEY; that there were "probative similarities" between the data file names and structures (which the court collectively refers to as "data structures") of the copyrighted program and those of the accused program; and that plaintiff *Baystate* had therefore shown "factual copying." (946 F.Supp. at 1086-87.) The court held, however, that plaintiff failed to prove substantial similarity because "the data structures at issue are not protected under the copyright laws, nor (even if they were so protected) are they constituent, original elements of the program." (*Id.* at 1087.)

Data structures are outside the subject matter of copyright. In a preliminary comment, but one which set the stage for the remainder of its analysis, the court observed that the Copyright Act defines a computer program as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result." (*Id.* at 1086) Accordingly, the court stated, the overall computer program comprising the allegedly infringed work is copyright protected. But "because the data structures at issue in this case do not bring about any result on their own, they are copyright protected, if at all, only as a part of the whole computer program." (*Id.*) Although the *Baystate* court made this preliminary statement regarding copyrightability of data structures, this was not the basis of its conclusion that the structures at issue in the case were not protectable. Instead, the actual holding of noncopyrightability clearly rested on the doctrines of merger and *scènes à faire*. The court held that the data file names were unprotected because merged with the underlying ideas or functions, and that the organization of the data files was unprotected as *scènes à faire*.

Merger. Under the merger doctrine, copyright protection is denied even to the expression of an idea if the underlying idea can be expressed in only a limited number of ways. When the possible ways of expressing an idea are so limited, the idea and the expression are said to have merged, and none of the "expressions" are considered copyrightable. *See, e.g., Apple Computer, Inc. v. Microsoft Corporation*, 35 F.3d 1435, 1444 (9th Cir. 1994); *Toro Company v. R&R Products Co.*, 787 F.2d 1208, 1212 (8th Cir. 1986); *Morrissey v. Procter & Gamble Co.*, 379 F.2d 675, 678-79 (1st Cir. 1967).

The merger doctrine, the *Baystate* court held, barred protection of the "data structure names or, more specifically, the words and abbreviations used to describe the files contained within the data structures and the data structures themselves." (*Id.* at 1088.) Although these "data structure names were independently created and were original expression," there was evidence that "the name of a file is typically related to its function." So, for example, a file that controls or creates color would typically have a name in which some form of the word "color" appears. The court therefore held the names unprotectable under the merger doctrine. (*Id.*) Although this holding appears case-specific on its face, in fact it would apply broadly to the terms which are an element of many technical interfaces, inasmuch as such function-invoking names or labels as system commands, parameter names and the like commonly use semi-descriptive, mnemonic terms for the practical purpose of making them easier to remember and work with.

The conclusion that such terms are merged and unprotectable is not unreasonable (though also not inevitable) under established doctrine. Two unexamined questions are posed by the Court's approach, however. First, is the proper unit of analysis for merger purposes the

7 It was probably for this reason that the principal of Baystate's predecessor in interest, the company originally known as Cadkey, Inc., freely provided program material needed to facilitate the creation of a product that could read CADKEY data files to the company that later began developing the translator for defendant Bentley. (*Baystate* at 1082-83.)

individual data structure name or the entirety of names? If the entirety were the appropriate subject of analysis, it would be more difficult to assert that the entire group of names is merged, for if there were even a small number of mnemonically-related, feasible alternative names for each data structure, taken together they would yield an exponentially large number of possible sets of names. However, this very way of describing the situation may suggest that the "entirety" is indeed merely a compilation of names, and does not reflect any kind of organic or unified whole. If the court was correct in judging the merger of the data structure names on an individual basis, the court's conclusion that the data structure names were (individually) merged naturally leads to the second unexamined question: Could not the collection of the data structure names be protected as a compilation?

Since the copyright in a compilation is circumscribed and limited to the originality shown in the selection and arrangement of such elements, *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 113 L.Ed.2d 358, 371 (1991), the protection would be narrow, and infringement would likely be judged under the "virtual identity" or "bodily appropriation" standard.[8] But this standard would not necessarily preclude a finding of infringement in this case, because the defendant likely did copy the entire body of data structure names.

Is the collectivity of data structure names a "selection" and/or "arrangement" within the meaning of copyright law on compilations? If an author assembles a collection of unprotected elements for some purpose and presents that collection in his work, the assemblage is only protectable as a compilation insofar as there is "expression" in "the manner in which the compiler has selected and arranged the [unprotected elements]." *Id*. In this context, if someone asserts that the "selection" of data structure names may qualify as a protectable compilation, that which is protected must be the act of judgment in deciding which unprotectably-named data structures to include within the compilation. In that event, there is a strong argument that that "selection" itself is unprotectable under the merger doctrine. The argument is that the selection of (merged and unprotected) names was itself merged with the substantive decision on which functional capabilities should be made available through the data structures. In other words, assuming *arguendo* (a big assumption) that all of the individual names are deemed unprotectable, all that is left to protect is the decision on which "names" to aggregate into the compilation. But that decision is the same as the functional decision on what structures and structural elements the author intends to include.

A similar issue arises with respect to whether there was a protectable "arrangement" of data structure names within the program. The overall array or arrangement of the data structure names is likely to reflect a logical ordering which may be entitled to "compilation" protection. But (at least with regard to the order in which the names are arrayed within any particular data structure), the arrangement of names can reasonably be said to be merged into the arrangement of substantive elements comprising the structure itself which the arrangement reflects. In other words, the programmer is likely to have exercised judgment in setting forth the arrangement of variables or other elements that make up a data structure or its sub-structures, or to have employed a syntax that may or may not be original. Having made those judgments and employed that syntax, the arrangement of the names given to those elements simply follows from (and should be deemed merged with) that underlying determination of the structure of data elements and sub-structures and the syntax employed.

Of course, this would not resolve the ultimate question of protectability, which turns on whether the structures themselves and their syntax (however they may be labeled) are protectable. Thus, if the underlying structures and syntax were deemed "ideas" or otherwise unprotected, the merger doctrine should preclude independent protection for the names themselves. If those elements were copyrightable, on the other hand, then the selection and arrangement of the associated data structure names as such should be protected as a compilation. Since the district court also held the underlying structures to be unprotectable, that we next turn to that issue.

Scènes à faire.

The doctrine known as "*scènes à faire*" excludes from copyright protection any "expressions" that are standard, stock, or common to a particular topic. In the genre of monster/disaster movies, for example, the story usually includes a handsome young scientist (accompanied by a beautiful female assistant) who devises a solution that rescues the planet just in time. Such formulaic elements are standard treatments, and anyone can use them. In the field of computer software, "standard programming techniques" and other common treatments (*e.g.*, a user interface in which a file is opened by double-clicking on an icon) are deemed unprotectable under the *scènes à faire* doctrine. This doctrine is also often used to deny protection to expression dictated by "external factors" such as (in the computer context) hardware standards and mechanical specifications, software

8 These are essentially synonymous terms for the level of similarity which is found when there is "copying or unauthorized use of substantially the entire item." *See, e.g., Apple Computer, Inc. v. Microsoft Corp.*, 821 F. Supp. 616, 623 (N.D. Cal. 1993), *aff'd*, 35 F.3d 1435, 1442 (9th Cir. 1994); *see also MiTek Holdings, Inc. v. Arce Engineering Co., Inc.*, 864 F. Supp. 1568, 1584 (S.D. Fla. 1994), *aff'd*, 89 F.3d 1548, 1558 (11th Cir. 1996) (where a work consists "largely of uncopyrightable elements," a standard of "virtual identicality" or "bodily appropriation" must be used to judge infringement).

standards and "compatibility" requirements, computer manufacturer design standards, target industry practices and demands, and computer industry programming practices. *See, e.g., Computer Associates Int' l, Inc. v. Altai, Inc.*, 982 F.2d 693, 709-10 (2d Cir. 1992); *Gates Rubber Co. v. Bando Chem. Indus., Ltd.*, 9 F.3d 823, 838 (10th Cir. 1993).

In *Baystate*, District Judge Gorton holds the data file structures unprotectable based on the *scènes à faire* doctrine, but engages in a sleight of hand to reach this result. The court states:

"Under the *scènes à faire* doctrine, protection is denied to those elements of a program that have been dictated by external factors. For computer programs, those external factors include, among other things, compatibility requirements and industry-wide programming practices. [¶] In this case, the court concludes that the selection and organization of the elements in the data files is dictated mainly by external factors." (P. 1088, citation omitted.)

The court can only reasonably be referring to the copyrighted program here, the program whose protectability is at issue, because the issue posed under the *scènes à faire* doctrine is whether "protection is denied" to elements of the copyrighted program because they were dictated by externalities. But in the sentences that immediately follow, Judge Gorton purports to justify his conclusion by reference to the accused program:

"The product being developed is a data translator that is designed to 'read' the data files of CADKEY. The process of 'reading' the CADKEY data files requires that the elements contained within the data structures of the Translator be organized in the same manner as the elements in the data structures of CADKEY." (*Id.*, emphasis added.)

Thus, the court asserts, the inability of the *translator* to function unless it is compatible means that the structures devised by the original author are unprotectable. (*Id.*) The court bolsters its conclusion that the structures are not copyrightable with further references to the needs of the copier: "[T]he organization of [file] names is, at least partly, a function of efficiency," because "the names and arrangement of those names serve a functional and necessary purpose *in the code of* a data translator." (*Id.*, emphasis added.)

The court's analysis confuses protectability with infringement or fair use. The fact that a translator must have the same data file structure as the program it is designed to interface with does not mean that externalities dictated these elements *in the original program*; contrary to the court's treatment in *Baystate*, the *scènes à faire* doctrine addresses whether features of *the original work* are denied protection because they were dictated in one important way or other. It simply represents a misreading and misunderstanding of the role of externalities to contend that the needs of a later competitor could mean that the original developer's authorship could be diminished on this ground.

A review of the cases which set forth the basis of the *scènes à faire* doctrine, in the context of computer programs, confirms that the core of the doctrine is the principle that an author cannot claim ownership of non-original "standard treatments" nor can she be permitted to effectively monopolize a market when the copyrighted work is expressed in a certain way because her program had to meet certain requirements and expectations of the customers of the program, or because she was mirroring a common programming practice, or because it had to be written in a certain way to be compatible with other programs or machines. It makes no sense to say that a program which was not originally "dictated" by such considerations becomes retrospectively dictated by externalities and is rendered unprotectable on the ground that later programs need to be compatible with the *first* program. The corpus of software copyright cases that elaborate the concept of *scènes à faire* do not involve such claims and do not contain such holdings.[9]

Industry standards

The court's invocation of "industry standards" likewise reflects a misunderstanding. The refusal to protect expression which reflects "accepted programming practices within the computer industry," *Computer Associates, supra*, 982 F.2d at 709-10, reflects the principle that no one can monopolize standard programming techniques. This is an entirely different matter, obviously, from asserting that certain

9 See, e.g., C*omputer Associates International, Inc. v. Altai, Inc.*, 982 F.2d 693, 609-10 (2nd Cir. 1992) (reviewing software copyright cases invoking *scenes a faire* doctrine and concluding "that a court must also examine the structural content of an allegedly infringed program for elements that might have been dictated by external factors"); *Atari v. Nintendo*, 975 F.2d 832, 839-40 (Fed. Cir. 1992) ("External factors did not dictate the design of the 10NES [allegedly infringed] program."); *Plains Cotton Co-op v. Goodpasture Computer Service*, 807 F.2d 1256, 1262 (5th Cir. 1987) (elements of a cotton market information program that are "dictated by the externalities of the cotton market" are not protectable); *Autoskill Inc. v. National Educational Support Sys.*, 994 F.2d 1476 (10th Cir.), *cert. denied*, 114 S. Ct. 307 (1993) (components of copyrighted program unprotectable under scenes a faire because they were "so standard" to the field of reading diagnosis and skills software); *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222, 1236 (3rd Cir. 1986) ("*Scenes a faire* are afforded no protection because the subject matter represented [by the first author] can be expressed in no other way than through the particular *scene a faire*" and a copyright would grant the first author a monopoly on commonplace ideas). There have been other cases, however, which have held either that the original author's domination of the market renders its practice with regard to some program element a *scéne á faire* and/or that the needs of subsequent authors are relevant to whether a treatment represents a *scéne á faire*. *See, e.g., Mitel Inc. v. Iqtel, Inc., supra*, 896 F.Supp. 1050.

types of program elements are unprotectable because it is the practice of others to copy such types of elements. Yet that was all the evidence the court in *Baystate* appeared to rely on in invoking "industry standards" as a basis for its *scènes à faire* holding: Although there was "relatively little evidence of the CAD industry standards," this limited evidence tended to show that those who developed translators had a practice of using the same data structure names and organization as that of the "target product" whose output was to be translated. There was apparently no evidence that *Baystate*'s actual data structures had become "industry standard"; at most, there was fragmentary evidence that the copying of data structures in general was an industry practice. This scarcely shows that *Baystate*'s authorship was constrained by industry practices, and is no ground for holding such elements unprotectable.

That the *scènes à faire* doctrine was misapplied by the court does not, of course, mean that the court came to the wrong conclusion regarding the protectability of *Baystate*'s data structures or regarding whether Bentley's use of those structures might be a "fair use" within the meaning of copyright law. Although, obviously, there are two ways of looking at the issue, there are plainly substantial arguments that a data structure—which is in essence a set of rules and principles that must be adhered to in order to create compatible files and to access data—is either an idea or a process, procedure or method of operation, within the meaning of § 102(b) of the Copyright Act.

The result here does not appear to be directly controlled by the *Borland* decision of the First Circuit, which focused heavily on the fact that the command hierarchy of Lotus 1-2-3 was merely the method (and the only method) by which end-users operated the program, and hence unprotectable. CADKEY end-users do not operate the CADKEY program by direct manipulation of the data structures, and it is unclear whether the First Circuit would extend its straightforward concept of "methods of operation" to include the "method" by which a program "operates" on data or files. But the First Circuit might well conclude, consistent with its approach in *Borland*, that *Baystate*'s data structures are processes or procedures; that all processes and procedures are unprotectable, however "expressed"; and that all comers are free to "build" upon *Baystate*'s structures without liability. To paraphrase *Borland*:

Although "in most contexts, there is no need to 'build' upon other people's expression," in "the context of [processes or procedures], 'building' requires the use of the precise [process or procedure] already employed; otherwise, 'building' would require dismantling, too. Original developers are not the only people entitled to build on the [processes or procedures] they create; anyone can." (*Borland* at 818.)

Infringement

The bulk of the *Baystate* court's copyright analysis was devoted to the issue of protectability, but the court briefly considered two further matters of interest.

First, the court, assuming for sake of argument that the data structures were protectable, addressed the issue of whether they were infringed, *i.e.*, whether the copying of data structures represented extensive enough copying to render the two works substantially similar. The court concluded that they were not sufficiently similar because data structures represented "neither a substantial portion nor a significant aspect of the whole copyrighted work." (*Id*. at 1089.) *Baystate*'s explanation of this conclusion is helpful, because it clarifies that the fact that a program element is necessary does not automatically mean that it is important enough to warrant a conclusion of infringement.

"[A]lthough data structures are generally a necessary component of a computer program for organizational and efficiency purposes, the original naming of the data structures takes very little of the total time or creative genius necessary to develop a program. Further, data structures are not, by themselves, executable . . .. Although the importance of a program component is not strictly a function of quantity," the evidence in *Baystate* demonstrated that the structures were only a small part of the total CADKEY program, and their copying would not render the works substantially similar. (*Id*. at 1090.)

Finally, the court turned to the issue of whether the creation of a program embodying *Baystate*'s data structures represented an infringement of *Baystate*'s documentation, which included an explanation of the data structure. The court held that it did not for two reasons. First, the translator source code was simply not, in the court's view, similar to the documentation in content, purpose or use. Second, under the Supreme Court's seminal case on what is sometimes referred to as "copying for use," *Baker v. Selden, 101 U.S.* 99 (1879), the copyright in a book does not grant an exclusive right to use the information conveyed in that work. Interestingly, the same argument could be made with respect to the header files that included and defined the data structure names and with respect to the set of files that embodied the structure—that is, it could be argued under *Baker v. Selden* that even if *Baystate* owns a copyright in those structures, that the copyright does not preclude use of the information contained in those files, and that Bentley's translator does not "copy" the structures (in a copyright sense) but merely "uses" them.

Conclusion

Although the reasoning of *Baystate v. Bentley* is not impeccable, the case squarely addresses the issue of the protectability of technical

interfaces and more clearly resolves the issue than any previous case the author is aware of. With the recent settlement of the case on appeal, the district court's decision in *Baystate* stands as an important contribution to the body of law on the subject.