No. 18-956

IN THE
Supreme Court of the United States

GOOGLE LLC,
Petitioner,

v.

ORACLE AMERICA, INC.,
Respondent.

On Writ of Certiorari to the United States Court of Appeals for the Federal Circuit

AMICUS CURIAE BRIEF OF DEVELOPERS ALLIANCE IN SUPPORT OF PETITIONER

BRUCE GUSTAFSON
DEVELOPERS ALLIANCE
1201 Wilson Blvd.
25th Floor
Arlington, VA 22209
(202) 735-7333
bruce@developersalliance.org

JAMES H. HULME
Counsel of Record
NADIA A. PATEL
ARENT FOX LLP
1717 K Street, NW
Washington, DC 20006
(202) 857-6000
james.hulme@arentfox.com

TABLE OF CONTENTS

Page

TABLE OF AUTHORITIES

Other Authorities

INTEREST OF *AMICUS CURIAE*

The Developers Alliance is a non-profit corporation that advocates for software developers.[1] Our corporate mission is to "[a]dvocate on behalf of developers and the companies that depend on them, support the industry's continued growth, and promote innovation."[2]

Alliance members include industry leaders in consumer, enterprise, industrial, and emerging software, and a global network of more than 75,000 developers.[3]

Amici have no direct financial interest in the outcome of this case but have a strong interest in seeing that the law continues to support innovation in the software industry. Due to the importance of the issues presented to the developer community, the Developers Alliance has been following this litigation closely. The Developers Alliance previously joined two *amicus* briefs in this matter before the Federal Circuit

---

[1] No counsel for any party authored this brief in whole or part, and no person other than *amicus curiae* or its counsel made a monetary contribution to the preparation or submission of this brief. All parties received timely notice of the Developers Alliance's intent to file and consented to the filing of this brief.

[2] https://www.developersalliance.org/about/about-the-alliance/.

[3] A list of Developers Alliance members is available at https://www.developersalliance.org/member-directory/. Google is a Developers Alliance member but took no part in the preparation of this brief.

and filed an *amicus* brief in support of certiorari in this proceeding.[4]

SUMMARY OF ARGUMENT

The current case has implications that go far beyond the two litigants involved. In 2017 there were an estimated three million software developers in the United States, and their collective work added an estimated $565 billion to the country's gross domestic product.[5,6] As a result of the current litigation, developers are now confused about whether and where established practices constitute copyright

---

[4] Brief of *Amici Curiae* Rackspace US, Inc., Application Developers Alliance, TMSoft, LLC, and Stack Exchange Inc., *Oracle America, Inc. v. Google Inc.*, Nos. 13-1021, 13-1022 (Fed. Cir. May 30, 2013); Brief of *Amici Curiae* Engine Advocacy, The App Developers Alliance, and Github Inc., *Oracle America, Inc. v. Google Inc.*, Nos. 17-1118, 17-1202 (Fed. Cir. June 1, 2017); Cert. Brief for Developers Alliance as *Amicus Curiae*.

[5] There are nearly three million professionals that are involved in software development and programming as part of their jobs. Over half of those are strictly software developers while the rest have occupations that require programming as a secondary component of their work such as computer scientists, data analysts, and database administrators. Developers Alliance & NDP Analytics, *Quantifying Risks to Interoperability in the Software Industry* (2017), https://www.developersalliance.org/interoperability-report-december-2017.

[6] In 2017, Software.org, the BSA Foundation, commissioned The Economist Intelligence Unit (EIU) to assess the economic impact of the software industry. The EIU collected and analyzed the most recent data available from several recognized and reputable sources. *The Growing $1 Trillion Economic Impact of Software* (2017), https://software.org/reports/2017-us-software-impact/.

infringement. Specifically, developers now question their ability freely to create interoperable software across projects and platforms, an established industry practice for decades. In determining whether copyright law should apply to API declarations, the Court should therefore consider the nature of software and industry norms.

The Court should also recognize that use and re-use of interoperable software is the foundation of innovation in the software development industry. Subjecting API declarations to copyright protections would force developers to constantly re-engineer something that already works, stymying creativity and innovation.

Finally, imposing restrictions on the use of API declarations is contrary to the goals of copyright law and the nature of software.

Accordingly, the Court should reverse the judgment of the court of appeals.

ARGUMENT

Technology has progressed since this Court decided *Baker v. Selden*, 101 U.S. 99 (1880). The nineteenth century saw great strides in the development of programmable machines, but the "software" and "hardware" of the age were predominantly paper tapes, punch cards and electromechanical systems. Since then, the ability to repurpose complex equipment without completely reinventing it has become a fundamental driver of innovation.

Software interfaces, or "API declarations," serve a similar function. When developers write original software (or "code"), there is a universal understanding that they hold protected rights in their work. To enable collaborative development and software interoperability however, developers must be free to connect their own code to remote code that other developers have written. This is commonly done through API declarations.[7] As used here, an API declaration is a short line of code that is part symbolic logic, part syntax, part symbolic notation, and part pseudo-English.[8] These statements are at the heart of the current case.

API declarations are used by software developers as shorthand for a remote block of software that performs a precisely defined function on behalf of the calling program. Through a single-line API declaration call, application developers avoid the need to embed the complete implementing code in their programs. Any program that exists today could conceptually be rewritten by removing all API declaration calls and replacing them with the entirety of the remote implementing code they represent (assuming the appropriate underlying hardware

---

[7] The acronym "API" is probably the most misused term in the long record of this case, and we use it here with great trepidation for fear of adding to the confusion. Such is the danger of trying to translate between two dialects (the exact point of this brief). In this brief, "API" is an umbrella term without clear definitional boundary—a concept, versus a thing. An API declaration is a thing (defined herein), distinct from implementing code (a different thing, and also defined).

[8] *See, e.g.*, App. to Pet. for Cert. 223a (the call "java.lang.Math.max").

environment). In this sense, API declarations are easily distinguished from implementing code; an API declaration's function is to simply stand-in-for and point-to the implementing code. The API declarations at the heart of this case exist in two places: 1) one-at-a-time in the application code that calls them, and 2) in libraries associated with the computer platforms that house the implementing code.

To communicate effectively to and from the remote implementing code, an API declaration must be precisely formatted and carry with it all the appropriate information for it to perform its function. In fact, there is only one correct API declaration format for any particular API, and any deviation renders it non-functional. Given the wide range of computational and control functions that can be appropriately off-loaded to APIs, the number and complexity of available APIs can be overwhelming for a developer. By convention, API declarations embody information that a skilled developer can use to categorize them and identify related APIs—an implicit roadmap that simplifies the developer's task of finding and remembering hundreds of APIs. An experienced developer can intuit the likely form of unknown but related APIs, and the overarching structure of the library used to hold them, by examining a small number of API declarations in context.

An "API declaration" as defined above would be easily understood by any developer, but the term itself is not used in the industry. The software developer community generally refers to the API declaration, the remote implementing code it refers

to, and the function that this code performs, using the singular term "API," relying on the context to provide a deeper meaning. In industry vernacular, an application developer that embeds an API declaration into their program is said to "call the API" (where API here refers to the remote implementing code that then "executes" or "runs" when the API declaration is encountered in the primary program). This loose mapping of terms is a primary obstacle to understanding how software interfaces should be treated under copyright law.

Copyright, applied to any specialized art, must take into account the unique nature of the domain as understood by those participating in the field.[9] A critical flaw in the record of this case has been a lack of a shared understanding by the courts of the vocabulary and basic practices involved in software development. It is easier to understand the fundamentals of what is being discussed if technology terms-of-art are used "as the natives use them," particularly where they overlap with words and concepts that carry alternative meanings in other contexts. Writing software is not the same as writing a novel; the lexicon, purpose, and industry norms are very different. What merits copyright protection in one field may not in the other.

---

[9] For instance, in its Circular 41, *Copyright Registration of Architectural Works*, the United States Copyright Office identifies "[i]ndividual standard features of the architectural work, such as windows, doors, or other staple building components" as unprotected by copyright—a non-obvious fact to an outsider, but uniquely relevant to the domain. https://www.copyright.gov/circs/circ41.pdf.

APIs are the established industry mechanism that promotes innovation and interoperability while protecting the creative works of individual developers. By publishing API declarations, developers enable their implementing code to interoperate with the code of other programmers. Without shared API declarations, there is no way to call an API and no way to identify what implementing code to execute, how to transfer the appropriate variables for the API to act on, or how to interpret the results when they are returned. Without shared API declarations, each device and program is an island, and modern software development simply cannot occur.

Interoperability through software interfaces increases innovation by allowing independent developers to build on the work of others. Interoperability allows for independent innovation in logically separate sections of a complex computer program by defining how information passes from one program section to another, and what actions will occur as a result. For instance, because of the use of APIs between the firmware of a mobile device, its operating system, and the applications software developers have written, consumers can freely add, delete and update apps without purchasing a new phone.[10] In fact, it is now easy for users to port their

---

[10] Traditionally, "hardware" refers to the physical aspects of a device, while "software" is the broad term for programs that run on hardware. "Firmware" is software that is semi-permanently placed in hardware (it might only be capable of modification in the factory, for instance), often forming part of the interface between the two.

entire collection of applications from one device to another. Interoperability also allows developers to specialize and thus creates efficiencies in the use of scarce programming skills. Finally, interoperability helps drive innovation by balancing the market power of the various participants in a complex software ecosystem.

If copyright law were to give control over both API declarations and implementing code to the original author, then software interoperability is a monopoly, controlled by the copyright owner, in frustration of copyright law's stated intent to promote innovation. API declarations are functional by their nature, and the conventions of the software industry that created them strictly limit how they can be structured. In any case, their role in the software development process is such that imposing copyright infringement for their use would be inappropriate. The use and re-use of computer code is foundational to the nature of the software development industry and has been since its inception.

The role of software interfaces to promote the independent development of interoperable software is acknowledged by copyright law. An independent application developer who wishes to write code that interoperates with the Java or Android platform *necessarily must* use the Java or Android API declarations. An independent platform developer who wishes to interoperate with Java or Android applications *necessarily must* re-use the Java or Android API declarations.

The Court should find that API declarations, unique to the nature of software, are not subject to copyright protection, or in the alternative that the free and open sharing of API declarations is protected as fair use. The judgment below in this case should therefore be reversed.

I. Copyright in Software Must be Viewed with the Nature of Software as Context.

Software developers have long lived in a world defined by the unforgiving logic of the underlying machine they seek to harness. The goal is simple: to make the machine do exactly what you want, and nothing that you do not want. For a software developer, ambiguity is an error, as is any other miscommunication with the underlying machine, which cannot intuit intention or overlook flawed instructions. Every time software developers write code, there follows a leap of faith as they execute the program to see whether the computer does what they intended. Finding and correcting errors—"debugging"—is a fundamental skill every programmer must master.

Once they have written and debugged a section of software, programmers are understandably reluctant to wade back in and change things. Even the re-keying of error-free code has the potential to introduce new errors. Worse still, changes in the code that surrounds the working segment can suddenly break the untouched, previously-functioning portion, because context matters, and information stored in computer memory changes as the program executes.

To help manage the software creation and error correction process, the computer industry adopted a philosophy of breaking down large computer programs into smaller inter-connected blocks, and segmenting computer systems into logical and physical subsystems to help isolate them from each other. This reduced the creation of new errors in old code, made errors easier to find and compartmentalize, and had the added benefit of allowing the re-use of software blocks that performed commonly used functions, like mathematical operations, logically sorting lists, or parsing strings of letters. At the same time, this segmentation process allowed multiple authors to focus their attention on each of the self-contained and interoperable blocks, rather than a single author having to manage the entire software program.

The philosophy behind this modular approach is fundamental to the nature of the computer industry. From the earliest computer systems, engineers have broken down larger problems into smaller parts, and redundancies have been identified and removed. Developers gained efficiencies by telling the computer to re-read previous portions of the written code rather than writing things twice. "Do" steps one, two and three, then "re-do" steps one and two, then do step four—instead of writing out steps one and two twice in the same program. Critical to this evolution was the ability to hand off from one block of code to another, and eventually from one author's code to another's. This ability is now the foundation of modern software development, so much so that software developers are unable to imagine a world that works any other way.

Given that computer functionality is the fundamental purpose of computer code, the interface rules amongst software and hardware elements must be universally understood and rigidly followed. Independent authors must be able to rely on remote authors to explicitly support software interface implementations that are absolutely compatible and strictly in keeping with agreed formats. To accomplish this, the specifics of reusable software interfaces must be shared knowledge amongst all industry participants.

As with any community, the nature of software development is shaped by shared knowledge, shared experience, and shared language. To the extent that many of the concepts in computing were first developed by engineers and scientists, the language of software is colored by technical terms and mechanical allusions. The complexity of computing systems means that virtually all efforts in the area are collaborative and require many individuals to work in concert with a shared understanding of the overall group activity. The resulting vocabulary borrows heavily from English, mathematics, and logic because each of these is already reflected in the underlying computer hardware, but it is decidedly not prose in the conventional sense.

Just as legal scholars can write a thesis on the nuanced meaning of "partner," software developers use words-of-art that defy easy definition. Thus, while developers write "code," the word has multiple, broad, and nuanced meanings that computer professionals understand, but that lay people struggle to comprehend. Similarly, though "API" was once an

acronym for Application Programming Interface, developers use it as a shorthand for many things and aspects of things that generally refer to the rules, syntax and tools that allow software blocks that perform some useful function to "connect" to other software that wants to make use of that function. Developers publish APIs, call APIs, write APIs and share APIs thousands of times a day.

The written language of programming—the superset of the various programming languages that also includes some universal norms and higher-level syntax—is also unique to the nature of the industry. It is a functional language that borrows terms and context from the other fields that influence science, such that elements like brackets and braces indicate both separation and grouping, for instance. Thus, letters such as "x" and "y" imply coordinate variables, but could be anything. And arbitrary names skew toward logical consistency with what they represent: "max" for a common statistical concept, for instance. The end result is a human readable meta-language where those skilled in the art can intuit what the symbols and words represent, and thus can puzzle out what the program is trying to do without actually running it on a computer. A knowledgeable developer can "read code" that a lay person cannot.

Introducing new terms into the developer lexicon is systematized and constrained by the functional bias inherent in the nature of software development. A new programming language that seeks to provide full programming flexibility must remain faithful to the bias towards utility that computer science embodies. "Max," "MAX," "maxi," or "aMax" could be

acceptable additions to a suite of APIs supporting statistical functions, since they imply a function and meaning; "Aardvark" as a mathematical function might be whimsical, but it would not be embraced since it is not anchored to any deeper concept and would result in code that was non-intuitive and hard to understand. The success of any platform or language depends on how easily developers can read, remember, and wield it, and ultimately how effective it is in accomplishing programming goals. It is therefore critical that the application of copyright respect the context in which software is developed.

In an attempt to bridge the specialized language of software development and the language of law and copyright, lower courts have turned to either analogy or translation. Analogies can be helpful by bringing familiar context to an alien field, but they tend to oversimplify and to bring with them implicit assumptions based on the model chosen. Similarly, translations seldom capture the nuance of a shared culture's understanding of its unique language.

Analogies from other fields tend either to over-emphasize or underemphasize the degree to which software interfaces are one-sided, two-sided, or "no-sided." Software interfaces, as already described, simultaneously separate, connect, and stand-in-the place-of blocks of code. As understood by developers, they are functional, rigidly formatted, and their exact syntax and function must be known by the independent developers on either side to enable interoperable software.

In the same vein, analogies other than the industry chosen "library" tend to imply greater or lesser creativity in the organization of the APIs they hold. Real-world libraries store and organize information based on a limited number of high-level characteristics such as author, title, subject or language. Their purpose is to simplify searching by those that know something, but not everything, about that for which they are looking. API libraries arose spontaneously in the early days of computer programming as the complexity of API documentation grew. API libraries make it easier to search for or call a particular API and understand its form and limitations, and they help developers understand the relationship between various APIs. In Java and Android, their organization is explicitly embedded in each API declaration. Reading an API declaration identifies where it is located in the library and how the library is structured. Conversely, the location of an API declaration in the library structure dictates much of its written form.

The syntax, symbology, and structure of an API declaration, while it may contain elements from English, mathematics, or logic, is a purely functional construct whose purpose is to reference the target implementing code without ambiguity. The structure and characters that make up the declaration aid a knowledgeable developer in intuiting the purpose of the target API and in identifying related APIs more easily. A skilled developer, knowing what mathematical or logical function is needed for a program to perform, should quickly and easily be able to find a suitable API in its associated library.

II. Consistent With the Goals of Copyright Law, Software Interfaces Simplify the Creation of Software and Promote Innovation.

Just as specialization of labor revolutionized the manufacturing sector, the ability to separate hardware and software into interoperable parts has revolutionized the technology industry. The key that has unlocked software innovation is the ability for software and hardware from many independent creators to interoperate with software and hardware from many other independent creators.

The size and complexity of software projects is growing steadily. A multi-player online game today can contain 5 million lines of code, while a luxury car can contain 100 million lines.[11] Software development no longer occurs only inside corporations but is now distributed both geographically and across time zones, with many independent developers contributing their effort and knowledge to a single project. The distribution of development effort is now so broad that any particular developer can expect to work on many software projects during their careers, often concurrently.

The next phase of technology evolution is already upon us. Fully interoperable devices are being linked together to form the internet of things (IoT). Household appliances, mobile phones, computers, wristwatches, doorbells and a vast array of sensors

---

[11] An informative chart with comparable code sizes for various technologies is available at https://www.visualcapitalist.com/millions-lines-of-code/.

and devices are already sharing information through a dynamic and evolving network of interfaces. Even more than in today's software industry, interoperability is fundamental to the nature of IoT. It is estimated that the global economic productivity resulting from IoT would drop by $77 billion over the next eight years if current interoperability practices were restricted.[12]

A. Software Interfaces Enable Parallel Innovation in Multiple Layers of Hardware/Software Systems and Accelerate Innovation by Enabling Multiple Developers to Collaborate Across Large Software Projects.

Software interfaces promote innovation by breaking complex software systems into large numbers of component parts that can be improved in parallel. Rather than a single author capable of limited creative iterations, software interfaces allow a multitude of authors to work simultaneously.

To support the tremendous demand for new software, the developer community has adopted a number of universal practices. First, developers rely on libraries of common software functions written by others, rather than recode these functions themselves for every project. This improves developer efficiency. It is also standard practice for developers working on

---

[12] Developers Alliance & NDP Analytics, *Quantifying Risks to Interoperability in the Software Industry* (2017), https://www.developersalliance.org/interoperability-report-december-2017.

large projects to coordinate the efforts of several individuals and create interoperable code modules that can be connected and reconnected to achieve the larger programming goal. This allows developers to specialize and to tackle large tasks as part of a community. Thirdly, industry has focused on isolating the underlying device hardware from the software above it by adopting a number of more standardized platforms that bridge the gap between general purpose software and proprietary hardware. This creates opportunities to develop software programs that run on many different devices without having to rewrite new code for each device. In all cases, the key enablers are software interfaces that connect code blocks and manage the controlled transfer of formatted information between layers and blocks of software.

Because the use of software interfaces drives such universal benefit, it is generally accepted in the developer community that these structures should be widely available and easy to implement. The result has been the emergence of the open-source software community to share code, repositories of published APIs, and the rise of software platforms like Java and Android to enable greater interoperability across a wide range of devices. It has also led to the emergence of reusable software development tools tailored to the most popular software languages. A developer who can master these tools gains efficiency and can then apply these gains to a wide range of projects. This in turn has led to competition amongst the software tool builders to capture a broad community of developers to increase the value of their programming platform—a virtuous cycle. Simply put, interoperability is a

deeply embedded principle in modern software development today.

B. Software Interfaces Promote Competition and Increase Innovation by Preventing Lock-In Between Otherwise Independent Layers in Hardware/Software Systems.

The interoperability of software systems hinges on the interfaces between the many component parts. Who has rights to these interfaces, and how those rights are allocated, is critical to the future of software development. This cannot be overstated.

If access to API declarations remains separate and independent of rights to the implementing code they represent, then the ability to collaborate and to create interoperable software and hardware will proceed at its current frenetic pace and the goals of intellectual property law will be met. The advantage of this arrangement is that it places no penalty on developers for sharing, and it encourages market competition by creating a mechanism for application developers to easily call on the comparable APIs of many competing implementing code authors. More effective implementing code gets re-used more often by the community, and thus the community as a whole produces better software. The reputation and prospects of the best authors rise accordingly, providing them economic benefits in their future work. If, on the other hand, the use and implementation of API declarations are subject to individual control, then these interfaces become a choke point for monopoly control.

Conventional practice is for the developer of a new API to "publish" the details of what functions their new program performs, what parameters it requires, and what limitations it might have. In addition, in order for other developers to call the API from their programs, the API author must publish the API declaration. The implementing code may or may not be published, but in either case industry acknowledges that copyright in the implementing code rests with the original author.

It is accepted in the developer community that the API declaration itself is separate from the implementing code. It is a unique, structured shorthand for any remote block of code capable of performing the named function consistent with the limitations and requirements of the original API's published description. The actual code that sits behind the API declaration is irrelevant to the calling program, so long as it accepts the appropriate inputs in the established form and performs the expected actions. The implementing code likewise needs no awareness of the overarching purpose of the application software that calls it; its role is to accept the rigidly formatted call, and to take the promised actions in keeping with its published function.

It is also accepted in the industry that developers are free to innovate on both sides of the API declaration, creating new, complex programs that call existing APIs, but also writing improved implementing code in competition with an original author. Of note, developers are not free to modify the API declaration itself, because its strictly defined structure and syntax are fundamental to its ability to

perform its function. An independent change to the API declaration by the author of implementing code on one side, or by a software developer seeking to embed the API function in the developer's own code, would break the link between the two blocks of code and render the entire API non-operational.

If API declarations are controlled by the author of the original implementing code, innovation would be seriously reduced. First, no one would be free to improve on existing implementing code without creating a new and unique API declaration—requiring all existing application software which called the old API to be updated. Second, developers would be restricted to software and hardware environments supported by the author of the API declarations they chose to use. In either case, the range of devices and services available to the public would be vastly reduced, and the interoperability of software and hardware would be severely impacted.

In addition, the process of licensing and allocating the rights and obligations tied to API declarations would emerge as an adjunct to software development. Application developers would need certainty as to whether an API author was committed to improving and maintaining their API, how they would work to promote its implementation in software tools and in a range of hardware environments, and what licensing terms would apply to the various ecosystem participants involved. All this would take time, energy, and investment that otherwise might go to software development and innovation.

III. Restricted Rights in API Declarations and Libraries are Inconsistent With Copyright Law and the Nature of Software.

Copyright law enshrines a software developer's rights to the "creative works" embodied in written software, as distinct from the functional elements of software, which are not protected under either copyright or patent law.[13] Wherever Congress has vested or restricted rights in a software developer's work it is to promote innovation, consistent with the Constitution.[14] Congress has specifically identified the promotion of software interoperability as a goal of copyright law.[15]

Restricting the rights of developers to the free and open use of API declarations restricts interoperability by giving the original author control over any future innovation by other developers on either side. Given that the relationship and character of and between APIs is inherent in their form, restricted rights in the structure of API libraries has a similar effect.

---

[13] 17 U.S.C. § 102(a) & (b).

[14] *See* U.S. Const. art 1, § 8, cl. 8 ("Congress shall have Power . . . To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their Writings and Discoveries . . . .").

[15] *See, e.g.*, 17 U.S.C. § 1201(f)(1) (carving interoperability out of the Digital Millennium Copyright Act).

A. Restrictions on API Declarations Would Frustrate the Purpose of Copyright Law, Because Free and Open APIs Promote Innovation.

Shared API declarations enable independent and parallel innovation in both the software that calls the API and in the implementing code that responds to the API call. Authors of application software that relies on an API can write code confident that, should a future author come up with a better implementation of the API in question, their original software can benefit without any modification. For creators of APIs, the ability to innovate by creating a better implementation of a popular API can have an immediate impact by improving the performance of all the applications that call it.

If API declarations were controlled by the original author of the implementing code, competing API authors would need to create unique API declarations of their own—even for APIs whose function was identical. In turn, application developers would need to re-write their code to swap out the old API declaration with the new one. In the worst case, where the new and old API are only selectively available in target operating environments, developers might need to call both APIs, and implement the code necessary to manage all possible errors and conflicts that could arise as one, the other, or both respond to the API call.

In either scenario above, each competing API would need to find a unique API declaration for the common function from a limited list of choices that

would satisfy the industry's bias towards simple and logically consistent terms and structures. Ultimately this would deter competition in implementing code and increase the effort for application developers to write code that was interoperable with many hardware environments.

The logical extension of applying copyright protection to API declarations would be an absolute monopoly on all implementing code written in the future. Were a platform developer to simply look at Java applications, read the API declarations they contain, and, in complete ignorance of the existence of the Java platform, create a new platform from the ground up which was responsive to the visible API declarations in application code, the platform developer would conceivably infringe the original author's copyright.

Developers invest heavily in mastering a particular programming language. The more places where a specific language can be used, and the more portable the resulting code is, the more valuable a language is to learn and to master. If someone builds a platform that allows this common language to be used efficiently and in many foreign contexts, then the language gains in popularity and its attractiveness increases. If the platform is later restricted however, developer investment in the language is frustrated and innovation is reduced.

Software developers invested in learning the Java programming language in reliance on the promise of

"write once, run anywhere."[16] At its heart, this promise is a commitment that Java API declarations—the software interface between "write" and "run"—would be unrestricted. To the industry, the promise was a library of free and open API declarations and a commitment to encourage platform adoption throughout the industry. That Java's authors separately licensed their implementing code as part of a platform was unremarkable and appropriate, as was Google's independent investment in its own re-implementation. But in no way did the industry anticipate that Java could later exert control over the interface they had published.

If API declarations can be owned and licensed, the value of the platform implementing code shifts from the cost to re-implement (like Android), to the cost to re-implement *and* to replace a developer community invested in the related tools (once the community is established). If, by extension, it becomes common for all languages and platforms to manufacture this value shift once their developer communities mature, then developers will limit their investment in any particular system, knowing its popularity is finite, and efficiency and innovation will suffer.

---

[16] App. to Pet. for Cert. 4a.

B. Copyright of API Declarations is Inconsistent With the Letter of Copyright Law, Because API Declarations are Functional and Not Creative.

Copyright protects a software developer's creative works.[17] To be adopted successfully in the developer community, API declarations must reflect some logical relationship with the function of the implementing code they call, otherwise they are hard to remember and create ambiguous and unreadable code. An enterprising developer could conceivably publish API declarations using every reasonable term that implies the "maximum value" statistical function and point them to a single version of implementing code, effectively cornering the market for this API. This may sound extreme, but such behavior is common elsewhere, as can be seen in the market for domain names.[18]

Merger is the principle that where an idea "can only be expressed in a limited number of ways," the means of expression "cannot be protected, lest one author own the idea itself."[19] The exact form of an API declaration is largely dictated by the overarching rules of the computer language being used, the structural and logical foundation of computer

---

[17] 17 U.S.C. § 102(a).

[18] *See Domain name speculation*, Wikipedia, https://en.wikipedia.org/w/index.php?title=Domain_name_speculation&oldid=928471617.

[19] *Zalewski v. Cicero Builder Dev., Inc.*, 754 F.3d 95, 102–03 (2d Cir. 2014).

programming, and the syntactic conventions of the API collection involved. In general, "[API declarations] communicate[] to programmers what each program does, how it relates to the other programs, and what you need to do to make it work."[20] Any variation from the accepted formula renders the API declaration non-functional. Any new API declaration must follow the rules of the languages and libraries where it hopes to be embedded.

Even an original author that builds a novel programming language and a suite of API declarations is constrained in the creativity the author can wield, given the established industry norms around programming terms generally. Elements which describe actions, functions, state, or relationships are drawn from existing English, logic and mathematics, and the computer languages that have gone before. Where novel terms and symbols are introduced, there is tremendous pressure for defensible logic in the choice.

That is not to say that computer science eschews creativity in its language. But the history of computer science clearly differentiates between non-functional terms (the "*Java*" language, for instance), and functional terms that map to the underlying concepts and operations that the computer is expected to perform.

In addition to restrictions on creativity in the form of API declarations, they are purely functional in nature. Copyright law does not protect functional

---

[20] Brief in Opposition 6.

works, or the functional aspects of otherwise creative works.[21,22] The sole purpose of an API declaration is to stand-in-for and call the implementing code. An API declaration is an operational line of code that directs the program to pass control and execution to a remote block of code before moving on to the next program step. An API declaration does nothing by itself to affect the application program's goals. From a developer's perspective, it is a placeholder for the remote implementing code that avoids recreating equivalent implementing code, and the associated operating environment, inside the calling program.

C. Absolute Restrictions on API Declarations and Libraries are Inconsistent with Established Exceptions to Copyright Under the Law.

The doctrine of fair use limits the exclusive rights that copyright provides when "rigid application of the copyright statute . . . would stifle the very creativity which that law is designed to foster."[23] The Copyright Act recognizes that enabling interoperability of

---

[21] 17 U.S.C. § 102(b).

[22] For example, the copyright office has determined that recipes are uncopyrightable. *See* U.S. Copyright Office, Circular 33, *Works Not Protected by Copyright* 2 (2017), https://www.copyright.gov/circs/circ33.pdf.

[23] *Campbell v. Acuff-Rose Music, Inc.*, 510 U.S. 569, 577 (1994); 17 U.S.C. § 107.

software is one of the law's goals.[24] The law also codifies a four-part test for determining fair use.[25]

In order for two software programs to interoperate, at least some minimal exchange of information is required—even to just acknowledge each other. Java and Android API declarations, apart from correct structure, symbology and syntax, include terms which map to the specific implementing code that is being addressed, along with any program variables required for the implementing code to act upon. The nature of software programming requires an exact match between the line of code included in the originating program, and what is expected by the remote program—any variation results in a program error. For this reason, API declarations are widely shared and diligently scripted within the programmer community, compiled, organized for easy retrieval, and published in computer-language-specific libraries for all to use.

The role of software interfaces to promote the independent development of interoperable software is acknowledged by copyright law.[26] Java and Android API declarations are necessary elements to achieve interoperability between application programs and the Java or Android implementing code. The only way to enable interoperability is to re-use them exactly. An independent application developer who wishes to write code that interoperates with the Java or

---

[24] *See* 17 U.S.C. 1201(f)(3).

[25] 17 U.S.C. § 107(1)–(4).

[26] *See* 17 U.S.C. § 107(2).

Android platform *necessarily must* use the Java or Android API declarations. An independent platform developer who wishes to interoperate with Java or Android applications *necessarily must* re-use the Java or Android API declarations. This is in the nature of software.

Section 107 includes as one of its factors "the nature of the copyrighted work."[27] Accordingly, courts should consider the various aspects of a creative work with reference to the field in which the work belongs.[28] For example, in *Andy Warhol Foundation for the Visual Arts, Inc. v. Goldsmith*, 382 F. Supp. 3d 312, 322–23 (S.D.N.Y. 2019), the court considered the "protectable, original elements of a photograph [which] include[d] 'posing the subjects, lighting, angle, selection of film and camera, evoking the desired expression, and almost any other variant involved.'" *Id.* at 322 (quoting *Rogers v. Koons*, 960 F.2d 301, 307 (2d Cir. 1992)). By contrast, aspects such as lighting, angle, camera selection, and evoking the desired expression would be irrelevant to copyright in the fields of literature, architecture, or software.

Notwithstanding the statutory language, courts have consistently minimized the second factor in

---

[27] *Id.*

[28] "To evaluate whether a particular use qualifies as 'fair use,' [the court] must engage in an 'open-ended and *context-sensitive* inquiry.'" *Swatch Grp. Mgmt. Servs. Ltd. v. Bloomberg L.P.*, 756 F.3d 73, 81 (2d Cir. 2014) (emphasis added) (quoting *Blanch v. Koons*, 467 F.3d 244, 251 (2d Cir. 2006)).

assessing fair use.[29] This factor may be of little importance in areas where the application of copyright is well understood, but copyright in software is still largely misinterpreted and conflated with copyright in prose or the written arts. Thus, the nature of software is critical because it is significantly different from that of literature, news reporting or architecture. The copying of dozens of paragraphs from a literary work may be a violation of copyright, while the copying of every window from one architectural design to another may not. Courts recognize the distinct nature of the two fields, and no court would analogize between the two. To dismiss this factor in a fair use analysis of software is an error courts should avoid.

The record in this case is replete with examples of the court equating software with ordinary prose. A court or a reasonable juror, aware of both literature and software, should easily be able to determine that the nature of the two is significantly different. For example, the role of capitalization, spacing, and punctuation in prose can be a creative element, while in software it is usually rigidly formulaic and highly functional. The challenge courts must overcome is to recognize that, despite their similarities, the different nature of software and prose is a significant factor under copyright. The application of fair use must recognize and accommodate this. API declarations are not names, or labels, or chapter headings, or slogans.

---

[29] "The second factor has rarely played a significant role in the determination of a fair use dispute." *Authors Guild v. Google, Inc.*, 804 F.3d 202, 220 (2d Cir. 2015) (citing William F. Patry, *Patry on Fair Use* § 4.1 (2015)).

Their role and composition have no parallel in prose. They are universally shared and fundamentally important for innovation in software development, past and future. Copyright should have no hold over them.

CONCLUSION

To award copyright protection in a software interface to a single author is inconsistent with copyright law, established industry practice, and wise public policy.

The judgment of the court of appeals should be reversed.

Respectfully submitted,

James H. Hulme
Counsel of Record
Nadia A. Patel
ARENT FOX LLP
1717 K Street, NW
Washington, DC 20006
(202) 857-6000
james.hulme@arentfox.com

Bruce Gustafson
Developers Alliance
1201 Wilson Blvd., 25th Floor
Arlington, VA 22209
(202) 735-7333
Bruce@DevelopersAlliance.org

Attorneys for Amicus Curiae
Developers Alliance