DOI:10.1145/3322094

Vinton G. Cerf

APIs, Standards, and Enabling Infrastructure

Application programming interfaces (APIs) are handy programming tools for describing how one program can access the functionality of another. In many cases, an API might manifest as the definition of a subroutine call, describing how to reference the subroutine (for example, its *name*) and what *parameters* the subroutine is expecting and in what format, and the nature and format of any returned parameters resulting from invoking the subroutine call. It is common for *libraries* of subroutines to be created and APIs to them standardized so that programmers using the library can exercise the functionality of the subroutines at need. Interestingly, the standardization of the APIs can lead to the property that a collection of software that relies on the library subroutines might run successfully on more than one, independently programmed library, as long as the APIs used are the same (that is, refer to the same names and use the same parametric specifications).

This aspect of APIs leads to the same kind of interoperability that the Internet Protocol has provided in the Internet. Many protocols that lie *above* the IP *layer* (for example, transmission control protocol, user datagram protocol, real-time protocol) can make use of a substantial array of alternative underlying network technologies (for example, Ethernet, multiprotocol label switching, asynchronous transfer mode switches, software-defined networks, frame relay, point-to-point radio, laser links, and so on) without knowing anything about how these underlying layers work. They "see" the Internet through the lens of the standard Internet Protocol with its standard packet format and its addressing structure.

The Internet Protocol is sometimes referred to as the *narrow waist* of the Internet Protocol architecture. Think of an hourglass with a narrow waist. Above are all the protocols that are encapsulated in standard Internet Protocol packets and below are all the possible carriers of the IP packets. An API has a similar property. Any number of programs can be written that call upon the functionality of the libraries or operating system functions referenced by the APIs. More than one library or operating system can be programmed to behave in accordance with the standardized APIs. The commonality of the API *enables* the alternative implementation of operating systems and libraries, permitting "mixing and matching" of the application programs and the independently produced libraries or operating systems supporting them.

The widespread sharing of common or standardized APIs confers rich opportunities for choices of operating system or library implementations for the programming of applications. Underlying hardware, library, and operating system implementations can host the same application software on a wide range of platforms. The investment in platform, operating system, and library software can yield widespread benefits to the application programs that can run on any of them. Applications don't have to be rewritten and users benefit from enhanced choices.

Widely used programming languages often define common subroutines to make programming more efficient. Programmers do not have to create all new subroutines for common functions, rather, they can refer to the common functions using common APIs. For this benefit to be widely realized and to avoid having to reprogram applications to refer to the same functions by different names and formats, adoption of standardized APIs induces portability of applications to run on multiple platforms. Of course, these same programming languages typically also allow programmers to define their own, idiosyncratic subroutines for functions that may be unique to a particular application.

The availability of common libraries referenced by a common set of APIs also has the benefit that the common libraries may receive additional scrutiny for reliability and security owing to widespread use. But the standard APIs also allow for distinctly programmed libraries or operating system functions that are matched to the underlying hardware platform and its peculiarities and capabilities that are invoked by a common set of APIs or system calls.

A powerful example of standardization is the so-called Portable Operating System Interface (POSIX)[a] standardized by the IEEE. Innumerable applications intended to run in the UNIX operating system environment can be made to run on many variations of UNIX that implement the POSIX API. This is but one of many examples of the powerful, *enabling* effect of adopting common APIs with standardized reference names and parametric format. C

a https://en.wikipedia.org/wiki/POSIX

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.