Application programming interface

From Wikipedia, the free encyclopedia

In computer programming, an **Application Programming Interface** (**API**) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer. An API may be for a web-based system, operating system, database system, computer hardware or software library. An API specification can take many forms, but often includes specifications for routines, data structures, object classes, variables or remote calls. POSIX, Microsoft Windows API, the C++ Standard Template Library and Java APIs are examples of different forms of APIs. Documentation for the API is usually provided to facilitate usage.

Contents

- 1 Purpose
- 2 Uses
  - 2.1 Libraries and frameworks
  - 2.2 Operating Systems
  - 2.3 Remote APIs
  - 2.4 Web APIs
- 3 Design
- 4 Release policies
  - 4.1 Public API implications
- 5 Documentation
- 6 Copyright controversy
- 7 Examples
- 8 See also
- 9 References
- 10 Further reading

Purpose

Just as a graphical user interface makes it easier for people to use programs, application programming interfaces make it easier for developers to use certain technologies in building applications. By abstracting the underlying implementation and only exposing objects or actions the developer needs, an API reduces the cognitive load on a programmer. While a graphical interface for an email client might provide a user with a button that performs all the steps for fetching and highlighting new emails, an API for file input/output might give the developer a function that copies a file from one location to another without requiring that the developer understand the file system operations occurring behind the scenes.[1]

Uses

Libraries and frameworks

An API is usually related to a software library. The API describes and prescribes the *expected behavior* (a specification) while the library is an *actual implementation* of this set of rules. A single API can have multiple implementations (or none, being abstract) in the form of different libraries that share the same programming

interface. The separation of the API from its implementation can allow programs written in one language to use a library written in another. For example, because Scala and Java compile to compatible bytecode, Scala developers can take advantage of any Java API.[2]

API use can vary depending on the type of programming language involved. An API for a procedural language such as Lua could primarily consist of basic routines to execute code, manipulate data or handle errors, while an API for an object-oriented language such as Java would provide a specification of classes and their class methods.[3][4]

Language bindings are also APIs. By mapping the features and capabilities of one language to an interface implemented in another language, a language binding allows a library or service written in one language to be used when developing in another language.[5][6] Tools such as SWIG and F2PY, a Fortran-to-Python interface generator, facilitate the creation of such interfaces.[7]

An API can also be related to a software framework: a framework can be based on several libraries implementing several APIs, but unlike the normal use of an API, the access to the behavior built into the framework is mediated by extending its content with new classes plugged into the framework itself. Moreover, the overall program flow of control can be out of the control of the caller and in the hands of the framework via inversion of control or a similar mechanism.[8][9]

Operating Systems

An API can specify the interface between an application and the operating system.[10] POSIX, for example, specifies a set of common APIs that aim to enable an application written for a POSIX conformant operating system to be compiled for another POSIX conformant operating system. Linux and Berkeley Software Distribution are examples of operating systems that implement the POSIX APIs.[11]

Microsoft has shown a strong commitment to a backward-compatible API, particularly within their Windows API (Win32) library, so older applications may run on newer versions of Windows using an executable-specific setting called "Compatibility Mode".[12]

An API differs from an application binary interface (ABI) in that an API is source code based while an ABI is binary based. For instance, POSIX provides APIs, while the Linux Standard Base provides an ABI.[13][14]

Remote APIs

Remote APIs allow developers to manipulate remote resources through protocols, specific standards for communication that allow different technologies to work together, regardless of language or platform. For example, the Java Database Connectivity API allows developers to query many different types of databases with the same set of functions, while the Java remote method invocation API uses the Java Remote Method Protocol to allow invocation of functions that operate remotely, but appear local to the developer.[15][16] Therefore, remote APIs are useful in maintaining the object abstraction in object-oriented programming; a method call, executed locally on a proxy object, invokes the corresponding method on the remote object, using the remoting protocol, and acquires the result to be used locally as return value. A modification on the proxy object will also result in a corresponding modification on the remote object.[17]

Web APIs

Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets. An API approach is an architectural approach that revolves around providing programmable interfaces to a set of services to different applications serving different types of consumers.[18] When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format. While "web API" historically has been virtually synonymous for web service, the recent trend (so-called Web 2.0) has been moving away from Simple Object Access Protocol (SOAP) based web services and service-oriented architecture (SOA) towards more direct representational state transfer (REST) style web resources and resource-oriented architecture (ROA).[19] Part of this trend is related to the Semantic Web movement toward Resource Description Framework (RDF), a concept to promote web-based ontology engineering technologies. Web APIs allow the combination of multiple APIs into new applications known as mashups.[20] In the social media space, web APIs have allowed web communities to facilitate sharing content and data between communities and applications. In this way, content that is created in one place can be dynamically posted and updated in multiple locations on the web.[21]

Design

The design of an API has significant impacts on its usability.[1] The principle of information hiding describes the role of programming interfaces as enabling modular programming by hiding the implementation details of the modules so that users of modules need not understand the complexities inside the modules.[22] Thus, the design of an API attempts to provide only the tools a user would expect.[1] The design of programming interfaces represents an important part of software architecture, the organization of a complex piece of software.[23]

Several authors have created recommendations for how to design APIs, such as Joshua Bloch,[24] Kin Lane,[25] and Michi Henning.[26]

Release policies

APIs are one of the most common ways technology companies integrate with each other. Those that provide and use APIs are considered as being members of a business ecosystem.[27]

The main policies for releasing an API are:[28]

- Private: The API is for internal company use only.
- Partner: Only specific business partners can use the API. For example, car service companies such as Uber and Lyft allow approved third party developers to directly order rides from within their apps. This allows the companies to exercise quality control by curating which apps have access to the API, and provides them with an additional revenue stream.[29]
- Public: The API is available for use by the public. For example, Microsoft makes the Microsoft Windows API public, and Apple releases its APIs Carbon and Cocoa, so that software can be written for their platforms.

Public API implications

An important factor when an API becomes public is its *interface stability*. Changes by a developer to a part of it—for example adding new parameters to a function call—could break compatibility with clients that depend on that API.[30]

When parts of a publicly presented API are subject to change and thus not stable, such parts of a particular API should be explicitly documented as *unstable*. For example, in the Google Guava library the parts that are considered unstable, and that might change in a near future, are marked with the Java annotation `@Beta`.[31]

A public API can sometimes declare parts of itself as *deprecated*. This usually means that such part of an API should be considered candidates for being removed, or modified in a backward incompatible way. Therefore, deprecation allows developers to transition away from parts of the API that will be removed or unsupported in the future.[32]

Documentation

API documentation describes what services an API offers and how to use those services, aiming to cover everything a client would need to know to use the API. Documentation is crucial for the development and maintenance of applications that use the API.[33] API documentation is traditionally found in documentation files, but can also be found in social media such as blogs, forums, and Q&A websites.[34] Traditional documentation files are often presented via a documentation system, such as Javadoc or Pydoc, that has a consistent appearance and structure. However, the types of content included in the documentation differs from API to API.[35] To facilitate understanding, API documentation can include description of classes and methods in the API as well as "typical usage scenarios, code snippets, design rationales, performance discussions, and contracts", but implementation details of the API services themselves are usually omitted. Restrictions and limitations on how the API can be used are also covered by the documentation. For example, documentation for an API function could note that its parameters cannot be null, or that the function itself is not thread safe.[36] Because API documentation is so comprehensive, it can be difficult for the writers to keep the documentation updated and for the users to read it carefully, potentially resulting in bugs.[37]

API documentation can be enriched with metadata information like Java annotations. This metadata can be used by the compiler, tools, and by the *run-time* environment to implement custom behaviors or custom handling.[38]

Copyright controversy

In 2010, Oracle Corporation sued Google for having distributed a new implementation of Java embedded in the Android operating system.[39] Google had not acquired any permission to reproduce the Java API, although a similar permission had been given to the OpenJDK project. Judge William Alsup ruled in the *Oracle v. Google* case that APIs cannot be copyrighted in the U.S, and that a victory for Oracle would have widely expanded copyright protection and allowed the copyrighting of simple software commands:

> To accept Oracle's claim would be to allow anyone to copyright one version of code to carry out a system of commands and thereby bar all others from writing their own different versions to carry out all or part of the same commands.[40][41]

In 2014, however, Alsup's ruling was overturned on appeal, though the question of whether such use of APIs constitutes fair use was left unresolved.[42]

In 2016, following a two-week trial, a jury determined that Google's reimplementation of the Java API constituted fair use, but Oracle vowed to appeal the decision.[43]

Examples

- ASPI for SCSI device interfacing
- Cocoa and Carbon for the Macintosh
- DirectX for Microsoft Windows
- EHLLAPI
- Java APIs
- ODBC for Microsoft Windows
- OpenAL cross-platform sound API
- OpenCL cross-platform API for general-purpose computing for CPUs & GPUs
- OpenGL cross-platform graphics API
- OpenMP API that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran on many architectures, including Unix and Microsoft Windows platforms.
- Server Application Programming Interface (SAPI)
- Simple DirectMedia Layer (SDL)

See also

- API testing
- API writer
- Calling convention
- Comparison of application virtual machines
- Common Object Request Broker Architecture (CORBA)
- Document Object Model (DOM)
- Double-chance function
- Foreign function interface
- Interface (computing)
- Interface control document
- List of 3D graphics APIs
- Name mangling
- Open API
- Open Service Interface Definitions
- Platform-enabled website
- Plugin
- RAML (software)
- Software Development Kit
- Web API
- XPCOM

References

1. Clarke, Steven (2004). "Measuring API Usability". *Dr. Dobb's*. Retrieved 29 July 2016.
2. Odersky, Martin; Spoon, Lex; Venners, Bill (10 December 2008). "Combining Scala and Java". *www.artima.com*. Retrieved 29 July 2016.
3. de Figueiredo, Luiz Henrique; Ierusalimschy, Roberto; Filho, Waldemar Celes. "The design and implementation of a language for extending applications" (PDF). *TeCGraf Grupo de Tecnologia em Computacao Grafica*. Retrieved 29 July 2016.
4. Sintes, Tony. "Just what is the Java API anyway?". *JavaWorld*. Retrieved 29 July 2016.
5. Emery, David. "Standards, APIs, Interfaces and Bindings". Acm.org. Retrieved 2016-08-08.
6. "Appendix A. Creating a language binding for cairo". Cairographics.org. Retrieved 2016-08-08.
7. "F2PY.org". F2PY.org. Retrieved 2011-12-18.
8. Fowler, Martin. "Inversion Of Control".
9. Fayad, Mohamed. "Object-Oriented Application Frameworks".
10. Lewine, Donald A. (1991). *POSIX Programmer's Guide* (PDF). O'Reilly & Associates, Inc. p. 1. Retrieved 2 August 2016.
11. West, Joel; Dedrick, Jason (2001). "Open source standardization: the rise of Linux in the network era" (PDF). *Knowledge, Technology & Policy*. Springer. **14** (2): 88–112. Retrieved 2 August 2016.
12. Microsogt (October 2001). "Support for Windows XP". Microsoft. p. 4. Archived from the original on 2009-09-26.
13. "LSB Introduction". Linux Foundation. 21 June 2012. Retrieved 2015-03-27.
14. Stoughton, Nick (April 2005). "Update on Standards" (PDF). USENIX. Retrieved 2009-06-04.

15. Bierhoff, Kevin (23 April 2009). "API Protocol Compliance in Object-Oriented Software" (PDF). *CMU Institute for Software Research*. Retrieved 29 July 2016.
16. Wilson, M. Jeff. "Get smart with proxies and RMI". *JavaWorld*. Retrieved 29 July 2016.
17. Henning, Michi; Vinoski, Steve (1999). "Advanced CORBA Programming with C++". Addison-Wesley. ISBN 978-0201379273. Retrieved 16 June 2015.
18. "API-fication" (PDF download). *www.hcltech.com*. August 2014.
19. Benslimane, Djamal; Schahram Dustdar; Amit Sheth (2008). "Services Mashups: The New Generation of Web Applications". *IEEE Internet Computing, vol. 12, no. 5*. Institute of Electrical and Electronics Engineers. pp. 13–15.
20. Niccolai, James (2008-04-23), "So What Is an Enterprise Mashup, Anyway?", *PC World*
21. Parr, Ben. "The Evolution of the Social Media API". *Mashable*. Retrieved 26 July 2016.
22. Parnas, D.L. (1972). "On the Criteria To Be Used in Decomposing Systems into Modules" (PDF). *Association for Computing Machinery*. Retrieved 8 August 2016.
23. Garlan, David; Shaw, Mary (January 1994). "An Introduction to Software Architecture" (PDF). *Advances in Software Engineering and Knowledge Engineering*. **1**. Retrieved 8 August 2016.
24. Bloch, Josh. "How to design a good API and why it matters" (PDF).
25. Lane, Kin (2016-03-14). "The Industry Guide to API Design" (PDF). Kin Lane via 3scale. Retrieved 2016-03-14.
26. Henning, Michi. "API: Design Matters".
27. de Ternay, Guerric (Oct 10, 2015). "Business Ecosystem: Creating an Economic Moat". *BoostCompanies*. Retrieved 2016-02-01.
28. Boyd, Mark. "Private, Partner or Public: Which API Strategy Is Best For Business?". *ProgrammableWeb*. Retrieved 2 August 2016.
29. Weissbrot, Alison (7 July 2016). "Car Service APIs Are Everywhere, But What's In It For Partner Apps? | AdExchanger". *ad exchanger*. Retrieved 2 August 2016.
30. Shi, Lin; Zhong, Hao; Xie, Tao; Li, Mingshu (2011). "An Empirical Study on Evolution of API Documentation" (PDF). *International Conference on Fundamental Approaches to Software Engineering*. Springer Berlin Heidelberg. Retrieved 22 July 2016.
31. "guava-libraries - Guava: Google Core Libraries for Java 1.6+ - Google Project Hosting". Code.google.com. 2014-02-04. Retrieved 2014-02-11.
32. Oracle. "How and When to Deprecate APIs". *Java SE Documentation*. Retrieved 2 August 2016.
33. Dekel, Uri; Herbsleb, James D. (May 2009). "Improving API Documentation Usability with Knowledge Pushing". *Institute for Software Research, School of Computer Science*. Carnegie Mellon University. Retrieved 22 July 2016.
34. Parnin, Chris; Treude, Cristoph (May 2011). "Measuring API Documentation on the Web" (PDF). *Web2SE*. Retrieved 22 July 2016.
35. Maalej, Waleed; Robillard, Martin P. (April 2012). "Patterns of Knowledge in API Reference Documentation" (PDF). *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*. Retrieved 22 July 2016.
36. Monperrus, Martin; Eichberg, Michael; Tekes, Elif; Mezini, Mira (3 December 2011). "What should developers be aware of? An empirical study on the directives of API documentation" (PDF). *Empirical Software Engineering*. **17** (6): 703–737. doi:10.1007/s10664-011-9186-4. Retrieved 22 July 2016.
37. Shi, Lin; Zhong, Hao; Xie, Tao; Li, Mingshu (2011). "An Empirical Study on Evolution of API Documentation" (PDF). *International Conference on Fundamental Approaches to Software Engineering*. Springer Berlin Heidelberg. Retrieved 22 July 2016.
38. "Annotations". Sun Microsystems. Retrieved 2011-09-30..
39. "Oracle and the End of Programming As We Know It". DrDobbs. 2012-05-01. Retrieved 2012-05-09.
40. "APIs Can't be Copyrighted Says Judge in Oracle Case". TGDaily. 2012-06-01. Retrieved 2012-12-06.
41. "Oracle America, Inc. vs. Google Inc." (PDF). Wired. 2012-05-31. Retrieved 2013-09-22.
42. Rosenblatt, Seth (May 9, 2014). "Court sides with Oracle over Android in Java patent appeal". *CNET*. Retrieved 2014-05-10.
43. "Google beats Oracle—Android makes "fair use" of Java APIs". *Ars Technica*. Retrieved 2016-07-28.

Further reading

- Taina Bucher (2013). "Objects of Intense Feeling: The Case of the Twitter API". *Computational Culture* (3). ISSN 2047-2390. argues that "APIs are far from neutral tools" and form a key part of contemporary programming, understood as a fundamental part of culture.

Categories: Technical communication | Application programming interfaces

---