Application programming interface

"API" redirects here. For other uses, see API (disambiguation).
For the MediaWiki (the software used by Wikipedia) API, see mw:API

In computer programming, an **application programming interface** (**API**) is a set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types. An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising the interface. A good API makes it easier to develop a program by providing all the building blocks. A programmer then puts the blocks together.

In addition to accessing databases or computer hardware, such as hard disk drives or video cards, an API can ease the work of programming GUI components. For example, an API can facilitate integration of new features into existing applications (a so-called "plug-in API"). An API can also assist otherwise distinct applications with sharing data, which can help to integrate and enhance the functionalities of the applications.

APIs often come in the form of a library that includes specifications for routines, data structures, object classes, and variables. In other cases, notably SOAP and REST services, an API is simply a specification of remote calls exposed to the API consumers.[1]

An API specification can take many forms, including an International Standard, such as POSIX, vendor documentation, such as the Microsoft Windows API, or the libraries of a programming language, e.g., the Standard Template Library in C++ or the Java APIs.

An API differs from an application binary interface (ABI) in that an API is source code-based while an ABI is a binary interface. For instance POSIX is an API, while the Linux Standard Base provides an ABI. [2] [3]

1 Uses

1.1 API in procedural languages

In most procedural languages, an API specifies a set of functions or routines that accomplish a specific task or are allowed to interact with a specific software component. This specification is presented in a human readable format in paper books or in electronic formats like eBooks or as man pages. For example, the math API on Unix systems is a specification on how to use the mathematical functions included in the math library. Among these functions there is a function, named sqrt(), that can be used to compute the square root of a given number.

The Unix command man 3 sqrt presents the signature of the function sqrt in the form:

SYNOPSIS #include <math.h> double sqrt(double X); float sqrtf(float X); DESCRIPTION sqrt computes the positive square root of the argument. ... RETURNS On success, the square root is returned. If X is real and positive...

This description means that sqrt() function returns the square root of a positive floating point number (single or double precision), as another floating point number.

Hence the API in this case can be interpreted as the collection of the include files used by a program, written in the C language, to reference that library function, and its human readable description provided by the man pages.

Similarly, other languages have procedural libraries; for example, Perl has dedicated APIs for the same mathematical task with built-in documentation available, which is accessible using the perldoc utility:

$ perldoc -f sqrt sqrt EXPR sqrt #Return the square root of EXPR. If EXPR is omitted, returns #square root of $_. Only works on non-negative operands, unless #you've loaded the standard Math::Complex module.

1.2 API in object-oriented languages

In its simplest form, an object API is a description of how *objects work* in a given object-oriented language – usually it is expressed as a set of classes with an associated list of class methods.

For example, in the Java language, if the class Scanner is to be used (a class that reads input from the user in text-based programs), it is required to import the java.util.Scanner library, so objects of type Scanner can be used by invoking some of the class' methods:

import java.util.Scanner; public class Test { public static void main(String[] args) { System.out.println("Enter your name:"); Scanner inputScanner = new Scanner(System.in); String name = inputScanner.nextLine();

System.out.println("Your name is " + name + "."); inputScanner.close(); } }

In the example above, methods nextLine() and close() are part of the API for the Scanner class, and hence are described in the documentation for that API, e.g.:

public String nextLine()

Advances this scanner past the current line and returns the skipped input...

Returns:

the line that was skipped

Throws:

NoSuchElementException - if no line found

IllegalStateException - if this scanner is closed

More generally, in object-oriented languages, an API usually includes a description of a set of class definitions, with a set of behaviors associated with those classes. This abstract concept is associated with the real functionality exposed, or made available, by the classes that are implemented in terms of class methods (or more generally by all its public components hence all public methods, but also possibly including any internal entity made public like: fields, constants, nested objects, enums, etc.).

The API in this case can be conceived of as the totality of all the methods publicly exposed by the classes (usually called the class *interface*). This means that the API prescribes the methods by which one interacts with/handles the objects derived from the class definitions.

More generally, one can see the API as the collection of all the *kinds* of objects one can derive from the class definitions, and their associated possible behaviors. Again: the use is mediated by the public methods, but in this interpretation, the methods are seen as a *technical detail* of how the behavior is implemented.

For instance: a class representing a Stack can simply expose publicly two methods push() (to add a new item to the stack), and pop() (to extract the last item, ideally placed on top of the stack).

In this case the API can be interpreted as the two methods pop() and push(), or, more generally, as the *idea* that one can use an item of type Stack that implements the behavior of a stack: a pile *exposing* its top to add/remove elements. The second interpretation appears more appropriate in the spirit of object orientation.

This concept can be carried to the point where a class interface in an API has no methods at all, but only behaviors associated with it. For instance, the Java and Lisp language APIs include the interface named Serializable, which is a marker interface that requires each class implementing it to behave in a serialized fashion. This does not require implementation of a public method, but rather requires any class that implements this interface to be based on a representation that can be *saved* (serialized) at any time.[lower-alpha 1]

Similarly the behavior of an object in a concurrent (multi-threaded) environment is not necessarily determined by specific methods, belonging to the interface implemented, but still belongs to the API for that Class of objects, and should be described in the documentation. [4]

In this sense, in object-oriented languages, the API defines a set of object behaviors, possibly mediated by a set of class methods.

In such languages, the API is still distributed as a library. For example, the Java language libraries include a set of APIs that are provided in the form of the JDK used by the developers to build new Java programs. The JDK includes the documentation of the API in JavaDoc notation.

The quality of the documentation associated with an API is often a factor determining its success in terms of ease of use.

1.3 API libraries and frameworks

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

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.[5][6]

1.4 API and protocols

An API can also be an implementation of a protocol.

When an API implements a protocol it can be based on proxy methods for remote invocations that underneath rely on the communication protocol. The role of the API can be exactly to hide the detail of the transport protocol. E.g.: RMI is an API that implements the JRMP protocol or the IIOP as RMI-IIOP.

Protocols are usually shared between different technologies (system based on given computer programming languages in a given operating system) and usually allow the different technologies to exchange information, acting as an abstraction/mediation level between the two different environments. Protocol hence can be considered *remote* APIs, *local* APIs instead are usually specific to a given

technology: hence an API for a given language cannot be used in other languages, unless the function calls are wrapped with specific adaptation libraries.

To enable the exchange of information among systems that use different technologies, when an API implements a protocol, it can prescribe a *language-neutral* message format: e.g. SOAP uses XML as a general container for the messages to be exchanged, similarly REST API can use both XML and JSON.

1.4.1 Object exchange API and protocols

An object API can prescribe a specific object exchange format that a program can use locally within an application, while an object exchange protocol can define a way to transfer the same kind of information in a message sent to a remote system.

When a message is exchanged via a protocol between two different platforms using objects on both sides, the object in a programming language can be transformed (marshalled and unmarshalled[7]) in an object in a remote and different language: so, e.g., a program written in Java invokes a service via SOAP or IIOP written in C# both programs use APIs for remote invocation (each locally to the machine where they are working) to (remotely) exchange information that they both convert from/to an object in local memory.

Instead when a similar object is exchanged via an API local to a single machine the object is effectively exchanged (or a reference to it) in memory: e.g. via memory allocated by a single process, or among multiple processes using shared memory, an application server, or other sharing technologies like tuple spaces.

1.4.2 Object remoting API and protocols

An object remoting API is based on a remoting protocol, such as CORBA, that allows remote object method invocation. 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.[8]

When remoting is in place, a modification on the proxy object corresponds to a modification on the remote object. When only an object transfer takes place, the modification to the local copy of the object is not reflected on the original object, unless the object is sent back to the sending system.

1.5 API sharing and reuse via virtual machine

Some languages like those running in a virtual machine (e.g. .NET CLI compliant languages in the Common Language Runtime (CLR), and JVM compliant languages in the Java Virtual Machine) can share an API. In this case, a virtual machine enables language interoperability, by abstracting a programming language using an intermediate bytecode and its language bindings. In these languages, the compiler performs just-in-time compilation or ahead-of-time compilation transforming the source code, possibly written in multiple languages, into its language-independent bytecode representation.

For instance, through the bytecode representation, a program written in Groovy or Scala language can use any standard Java class and hence any Java API. This is possible thanks to the fact both Groovy and Scala have an object model that is a superset of that of the Java language; thus, any API exposed via a Java object is accessible via Groovy or Scala by an equivalent object invocation translated in bytecode.

On the other side, Groovy and Scala introduce first-class *entities* that are not present in Java, like closures. These entities cannot be natively represented in Java language (Java 8 introduced the concept of lambda expression); thus, to enable interoperation, a closure is encapsulated in a *standard* Java object. In this case the *closure invocation* is mediated by a method named call(), which is always present in an closure object as seen by Java, and in Java the closure does not represent a first-class *entity*.

2 Web APIs

Main article: Web API

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.[9] 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).[10] 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.[11]

2.1 Web use to share content

The practice of publishing APIs has allowed web communities to create an open architecture for 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:

- Photos can be shared from sites like Flickr and Photobucket to social network sites like Facebook and MySpace.
- Content can be embedded, e.g. embedding a presentation from SlideShare on a LinkedIn profile.
- Content can be dynamically posted. Sharing live comments made on Twitter with a Facebook account, for example, is enabled by their APIs.
- Video content can be embedded on sites served by another host.
- User information can be shared from web communities to outside applications, delivering new functionality to the web community that shares its user data via an open API. One of the best examples of this is the Facebook Application platform. Another is the Open Social platform.[12]
- If content is a direct representation of the physical world (e.g., temperature at a geospatial location on earth) then an API can be considered an "Environmental Programming Interface" (EPI). EPIs are characterized by their ability to provide a means for universally sequencing events sufficient to utilize real-world data for decision making.

3 Implementations

The POSIX standard defines an API that allows writing a wide range of common computing functions in a way such that they can operate on many different systems (Mac OS X, and various Berkeley Software Distributions (BSDs) implement this interface). However, using this requires re-compiling for each platform. A **compatible API,** on the other hand, allows compiled object code to function with no changes to the system that *implements that* API. This is beneficial to both software providers (where they may distribute existing software on new systems without producing and →distributing upgrades) and users (where they may install older software on their new systems without purchasing upgrades), although this generally requires that various software libraries implement the necessary APIs as well.

Microsoft has shown a strong commitment to a backward compatible API, particularly within their Windows API (Win32) library, such that older applications may run on newer versions of Windows using an executable-specific setting called "Compatibility Mode".[13]

Among Unix-like operating systems, there are many related but incompatible operating systems running on a common hardware platform (particularly Intel 80386-compatible systems). There have been several attempts to standardize the API such that software vendors may distribute one binary application for all these systems; however, to date, none of these has met with much success. The Linux Standard Base is attempting to do this for the Linux platform, while many of the BSD Unixes, such as FreeBSD, NetBSD, and OpenBSD, implement various levels of API compatibility for both backward compatibility (allowing programs written for older versions to run on newer distributions of the system) and cross-platform compatibility (allowing execution of foreign code without recompiling).

4 API design

Several principles are commonly used to govern the process of designing APIs. Parnas proposed the concept of information hiding in 1972. The principle of information hiding is that one may divide software into modules, each of which has a specified interface. The interfaces hide the implementation details of the modules so that users of modules need not understand the complexities inside the modules. These interfaces are APIs, and as a result, APIs should expose only those module details that clients must know to use modules effectively. Software architecture is dedicated to creating and maintaining high-level software structures—which typically includes modules. APIs reflect interfaces between modules. Thus, a system architecture is inextricably related to the APIs that express that architecture. However, many decisions involved in creating APIs are not architectural, such as naming conventions and many details on how interfaces are structured.

These details of how interfaces are structured, as well as the software architecture, have significant impacts on software quality. For example, Cataldo et al. found that bugginess is correlated with logical and data dependencies in software.[14] This implies that to reduce bug rates, software developers should carefully consider API dependencies.

Conway's Law states that the structure of a system inevitably reflects the structure of the organization that created it. This suggests that to understand how APIs are designed in the real world, one must also understand the structures of software engineering organizations. Likewise, an API group should structure itself according to what the API needs. In a study of 775 Microsoft software engineers, Begel et al. found that in addition to coordinating regarding API design, software engineers even more commonly coordinate regarding schedules and features.[15] This reinforces the view that software orga-

nizations collaborate extensively and that organizational structure is important.

Several authors have created recommendations for how to design APIs, such as Joshua Bloch[16] and Michi Henning.[17] However, since one of the principles of API design is that an API should be consistent with other APIs already in use in the system, the details of API design are somewhat language- and system-dependent.

5 Release policies

The main policies for releasing an API are:

- Protecting information on APIs from the general public. For example, Sony used to make its official PlayStation 2 API available only to licensed PlayStation developers. This enabled Sony to control who wrote PlayStation 2 games. This gives companies quality control privileges and can provide them with potential licensing revenue streams.
- Making APIs freely available. 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.

A mix of the two behaviors can be used as well.

5.1 Public API implications

An API can be developed for a restricted group of users, or it can be released to the public.

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.

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.[18]

5.2 API deprecation

A public API can sometimes declare parts of itself as *deprecated*. This usually means that such part of an API should be considered candidated for being removed, or modified in a backward incompatible way.

When adopting a third-party public API, developers should consider the deprecation policy used by the producer of that API; if a developer publicly releases a solution based on an API that becomes deprecated, he/she might be unable to guarantee the provided service.

6 API documentation

Professional-level documentation[19] for an API should strive to include the following parts:

Reference documentation A description of the functions and objects in the API (see the subsection API reference documentation)

Overview and concepts A narrative description of the different parts of the API and how they interact. Major frameworks in the API, such as its GUI, network, and file system frameworks should have their own separate section.

Tutorials/training classes Step-by-step instructions that show developers how to accomplish a particular task. The text should include code that developers can copy into their own applications. For example, a training class for a cryptographic API would include code that shows developers how to use the API to encrypt a file.

Installation/getting started/troubleshooting documentation One or more documents that show developers how to do the following:

- Obtain the software development kit (SDK) for the API
- Install the SDK on a development machine
- Obtain keys, accounts, and so forth that allow access
- Deploy or provide client libraries
- Troubleshoot problems with using the SDK

SDK tools documentation Documents that describe how to install and use build, compile, and deploy tools

License information Documents that describe the API license

6.1 API reference documentation

The reference documentation for an API is an intrinsic part of any API, and without it the API is unusable. Every aspect of the API, no matter how trivial, should be stated explicitly.

When an API documents a library of functions in a procedural language it should include:

- a description of all the data structures it depends upon
- a description of all the functions signatures, including:
  - function names
  - function parameters names (when it applies) and types
  - return type for the functions
  - for each parameter if the parameter is possibly subjected to modification inside the function
  - a description of the handling of any error condition
  - pre- and post-conditions or invariants
  - more generally how the *state* has changed after the function execution
  - possible side-effects
- any accessibility or visibility constraint.

An object API should document:

- the relationship of any type to other types: inheritance (super-types, sub-types, implemented interfaces or traits), composite structures, delegating entities or any mixed-in set of functionality
- the *public* part of an object derived from a class definition, hence:
  - its public constants
  - the name and type of the member variables (fields or properties) that are directly accessible for any object
  - the signature of the class methods including information similar to that for functions in procedural languages, possibly including a list of *getter* and *setter* methods used to access or modify encapsulated information
  - any class-specific operators, in case the language supports operator overloading
  - indication whether the fields or methods have a static nature
- any constraint that applies to the objects one can create
- nested structures, like inner classes or enumerations.

An API in a language using exception handling should report any kind of exception possibly thrown and the specific condition that can cause them to happen.

An API that can be used in a concurrent environment should include indications on how its behavior changes due to possible concurrent access to it: general usability in a concurrent context and possible race conditions.[4]

An API with unstable parts should document them as unstable.

An API with deprecated parts should document them as deprecated.

An API that implements a communications protocol should indicate its general behavior, and should detail:

- How to set up a communication session based on that protocol, and prerequisites for correctly setting up a communication session
- If the communication is stateful or stateless
- In case of stateful sessions: how to handle the *state*
- The *notation* for the kind of messages the protocol can transport
- How the protocol handles communication errors
- If, in case of communication errors, the protocol can resubmit a message
- Security levels supported, and how to secure communication
- Authentication required to set up a session
- If the communication can be associated to a transactional processing, and consequently how to handle transactions
- If the communication can be embedded in an extended conversation, and consequently how to handle the conversation

A graphical API should document:

- Graphical elements it can handle
- How to render graphical elements
- How to lay out elements on the graphical canvas, and how to compose them
- How to interact with graphical elements
- How to handle user input, e.g.,
  - How to add callback to specific user events
  - How to read information from input fields

An API that interacts with a device should document how to:

- Access the device to extract data from it
- Modify the state of the device, when possible
- Detect error conditions in the device.

An API should always indicate, where applicable:

- Language version number
- Library and other resource dependencies
- Protocol versions it is compatible with or that it implements
- Operating system or platform version it supports

An API that can be used in multiple languages via some form of language inter-operation should document any restrictions to its use by languages other than its *native* language.

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

7 APIs and copyrights

Main article: Oracle America, Inc. v. Google, Inc.

In 2010, Oracle sued Google for having distributed a new implementation of Java embedded in the Android operating system.[20] 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.[21][22]

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.[23]

2013 saw the creation of the "API Commons" initiative.[24] API Commons is a common place to publish and share your own API specifications and data models in any format such as Swagger, API Blueprint or RAML, as well as to explore and discover the API designs of others. The API specifications and data models declared in API Commons are available publicly under the Creative Commons license.

8 API examples

See also: Category:Application programming interfaces

- 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)

9 Language bindings and interface generators

APIs that are intended to be used by more than one high-level programming language often provide, or are augmented with, facilities to automatically map the API to features (syntactic or semantic) that are more natural in those languages. This is known as language binding, and is itself an API. The aim is to encapsulate most of the required functionality of the API, leaving a "thin" layer appropriate to each language.

Below are listed some interface generator tools that bind languages to APIs at compile time:

- SWIG – an open-source interfaces bindings generator supporting numerous programming languages
- F2PY – a Fortran to Python interface generator[25]

10 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 control document
- List of 3D graphics APIs
- Name mangling
- Open Service Interface Definitions
- Platform-enabled website
- Plugin
- Software Development Kit
- XPCOM
- RAML (software)

11 Notes

[1] This is typically true for any class containing simple data and no link to external resources, like an open connection to a file, a remote system, or an external device.

12 References

[1] "Customer Information Manager (CIM)" (PDF). *SOAP API Documentation*. Authorize.Net. July 2013. Retrieved 2013-09-27.

[2] "LSB Introduction". Linux Foundation. 21 June 2012. Retrieved 2015-03-27.

[3] Stoughton, Nick (April 2005). "Update on Standards" (PDF). USENIX. Retrieved 2009-06-04.

[4] Bloch, Joshua (2008). "Effective Java (2nd edition)". Addison-Wesley. pp. 259–312. ISBN 978-0-321-35668-0.

[5] Fowler, Martin. "Inversion Of Control".

[6] Fayad, Mohamed. "Object-Oriented Application Frameworks".

[7] "Java Platform, Enterprise Edition - The Java EE Tutorial - Release 7" (PDF). Oracle Corporation. 2014. para. 31.7 Using JAX-RS with JAXB. E39031-01. Retrieved 16 June 2015.

[8] Henning, Michi; Vinoski, Steve (1999). "Advanced CORBA Programming with C++". Addison-Wesley. ISBN 978-0201379273. Retrieved 16 June 2015.

[9] http://www.hcltech.com/sites/default/files/apis_for_dsi.pdf

[10] 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.

[11] Niccolai, James (2008-04-23), "So What Is an Enterprise Mashup, Anyway?", *PC World*

[12] "OpenSocial API Documentation". *Google Code*. Google. Retrieved 2007-11-02.

[13] Microsoft (October 2001). "Support for Windows XP". Microsoft. p. 4.

[14] Cataldo, M.; Mockus, A.; Roberts, J. A.; Herbsleb, J. D. (2009). "Software Dependencies, Work Dependencies, and Their Impact on Failures". *IEEE Transactions on Software Engineering* **99**: 864–878.

[15] Begel, Andrew; Nagappan, Nachiappan; Poile, Christopher; Layman, Lucas. "Coordination in Large-Scale Software Teams". *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*: 1–9.

[16] Bloch, Josh. "How to design a good API and why it matters" (PDF).

[17] Henning, Michi. "API: Design Matters".

[18] "guava-libraries - Guava: Google Core Libraries for Java 1.6+ - Google Project Hosting". Code.google.com. 2014-02-04. Retrieved 2014-02-11.

[19] Bisso, James; Maki, Victoria (2006). "Documenting APIs: Writing Developer Documentation for Java APIs and SDKs". Bitzone. ISBN 978-0963002105. Retrieved 16 June 2015.

[20] "Oracle and the End of Programming As We Know It". DrDobbs. 2012-05-01. Retrieved 2012-05-09.

[21] "APIs Can't be Copyrighted Says Judge in Oracle Case". TGDaily. 2012-06-01. Retrieved 2012-12-06.

[22] "Oracle America, Inc. vs. Google Inc." (PDF). Wired. 2012-05-31. Retrieved 2013-09-22.

[23] Rosenblatt, Seth (May 9, 2014). "Court sides with Oracle over Android in Java patent appeal". *CNET*. Retrieved 2014-05-10.

[24] "API Commons". API Commons. Retrieved 2014-02-21.

[25] "F2PY.org". F2PY.org. Retrieved 2011-12-18.

13 Text and image sources, contributors, and licenses

13.1 Text

- **Application programming interface** *Source:* https://en.wikipedia.org/wiki/Application_programming_interface?oldid=673710319 *Contributors:* Damian Yerrick, Sodium, Lee Daniel Crocker, CYD, Eloquence, Mav, Uriyan, Bryan Derksen, The Anome, Ed Poor, Andre Engels, Sfmontyo, Miguel~enwiki, Alan_d, Ellmist, Hfastedge, Frecklefoot, Bdesham, Danja, Michael Hardy, Wapcaplet, Ixfd64, Zeno Gantner, Skysmith, Mac, Nanshu, JWSchmidt, Mattknox, Andres, Denny, GRAHAMUK, Ehn, Hashar, Jengod, Robneild, Jitse Niesen, Quux, Tpbradbury, Itai, Bevo, Spikey, Bearcat, Robbot, Chealer, Gak, Mountain, RedWolf, ZimZalaBim, Altenmann, Peak, Hadal, Wikibot, Jleedev, Pengo, David Gerard, Enochlau, Ancheta Wis, Giftlite, JamesMLane, Darklight, Elf, Sj, Kim Bruning, Lupin, Dawidl, Chameleon, Uzume, Edcolins, Lockeownzj00, Szajd, Cynical, Funvill, Julien~enwiki, Syvanen, Ta bu shi da yu, Slady, Adah1972, Discospinster, Rich Farmbrough, Notinasnaid, Shlomif, Bender235, Limbo socrates, RJHall, CanisRufus, Freakimus, Aude, Diomidis Spinellis, RoyBoy, Bobo192, Smalljim, Redlentil, Enric Naval, Morg~enwiki, Colonoh, Apoltix, K12u, Scott Ritchie, Minghong, Obradovic Goran, Sam Korn, Chicago god, Anteru, Espoo, Atheken, M5, Zachlipton, AaronL, Poweroid, Diego Moya, Calton, Kocio, InShaneee, Loshsu, Wtmitchell, Velella, Suruena, Runtime, Pgimeno~enwiki, CloudNine, GeoffPurchase, Kbolino, Edwardkerlin, Skeejay, Thryduulf, Rocastelo, Ae-a, Robert K S, Sega381, Blackcats, Ivan007, TrentonLipscomb, Marudubshinki, King of Hearts (old account 2), SqueakBox, Graham87, Qwertyus, Yurik, DeadlyAssassin, MZMcBride, SMC, Ronocdh, Troymccluresf, Kichik, FlaBot, Riluve, Tijuana Brass, SteveBaker, Ahunt, Salvatore Ingala, Chobot, Bgwhite, YurikBot, Wavelength, Borgx, NTBot~enwiki, Stephenb, Manop, ENeville, Heraclius, Dipskinny, Debot~enwiki, Daniduc, JulesH, Davemck, Voidxor, Tony1, Alex43223, Bota47, Jmalin, Plamka, Cedar101, Bryanmonroe, JLaTondre, Teryx, Rwwww, GrinBot~enwiki, SmackBot, Dqpeck, Mihai cartoaje, KnowledgeOfSelf, Unyoyega, Fitch, Eskimbot, Brianski, Ohnoitsjamie, Hmains, Thumperward, DHN-bot~enwiki, Asydwaters, Zhinz, Frap, Addshore, Cybercobra, T-borg, EVula, Dreadstar, BryanG, Miohtama, Myc2001, Dmh~enwiki, Derek farn, Harryboyles, Jidanni, Mwarf, Soumyasch, Mbeychok, RichMorin, JHunterJ, SQGibbon, TastyPoutine, MTSbot~enwiki, Phuzion, Pwforaker, Wjejskenewr, Aeternus, Aeons, Courcelles, Jbolden1517, Atreys, OlegMarchuk, Kickin' Da Speaker, WeggeBot, HenkeB, Old Guard, Neelix, DShantz, Andkore, Yaris678, Michaelas10, ST47, Farrwill, Pascal.Tesson, Miketwardos, Chiefcoolbreeze, Thijs!bot, Epbr123, Davron, Humu, Muaddeeb, Escarbot, Hires an editor, AntiVandalBot, Seaphoto, Minirogue, MoreThanMike, Avk15gt, Spencer, Deadbeef, JAnDbot, XyBot, Husond, Deepugn, Greensburger, VoABot II, SHCarter, Tedickey, Torchiest, Hbent, Stephenchou0722, R'n'B, Onixz100, Mange01, Ahzahraee, Jesant13, Drewmeyers, Looc4s, WTRiker, SlowJog, FrummerThanThou, Raise exception, NewEnglandYankee, Rktur, DeeKay64, DorganBot, Jtowler, Izno, Infinitycomeo, Mpbaumg, Randomalious, SirSandGoblin, VolkovBot, Thisisborin9, Mrh30, JohnBlackburne, Tseay11, Philip Trueman, TXiKiBoT, Oshwah, Rponamgi, Dschach, Rei-bot, Qxz, Michael Hodgson, Steven J. Anderson, Econterms, Jackfork, Quinet, Daperata, YordanGeorgiev, Legoktm, Renatko, SieBot, John.n-irl, Kernel Saunters, Jerryobject, Bentogoa, Lavers, Nopetro, Aruton, Oxymoron83, SimonTrew, OKBot, Tantrumizer, Loren.wilton, ClueBot, PipepBot, Awg1010, The Thing That Should Not Be, WaltBusterkeys, Wysprgr2005, Boing! said Zebedee, HUB, Jmclaury, LizardJr8, Dylan620, Liempt, Auntof6, DragonBot, Excirial, Lartoven, Willyos, Rudyray, Sun Creator, Arjayay, Amanuse, Teutonic Tamer, TheresaWilson, BOTarate, Andy16666, Johnuniq, Egmontaz, DumZiBoT, XLinkBot, Kurdo777, Bikingviking, WikHead, Zodon, Airplaneman, Dsimic, Varworld, Pearll's sun, Sakhal, Betterusername, Landon1980, Queenmomcat, Fieldday-sunday, Reemrevnivek, Mac Dreamstate, MrOllie, Download, LaaknorBot, Ginosbot, Tide rolls, Zorrobot, Jarble, Aarsalankhalid, Luckas-bot, Yobot, Ptbotgourou, Fraggle81, Xqt, Scohil, Dmarquard, AnomieBOT, Efa, Piano non troppo, 90, ChristopheS, Materialscientist, Leoholbel, Mquigley8, Chadsmith729, Jpe.pinho, Vkorpor, Ffffloyd, ArthurBot, LilHelpa, Xqbot, Vikramtheone, Capricorn42, Liorma, TechBot, Rs rams, Ohspite, Boyprose, Utype, Joeldippold, Transpar3nt, Nichobot, RibotBOT, Slurrymaster, IShadowed, Shadowjams, Ebessman, Prari, FrescoBot, Omniscientest, Pepe.agell, Zero Thrust, DivineAlpha, Andremi, Dr Marcus Hill, PaymentVision, Rackspacecloud, Gryllida, Martyn Lovell, Lotje, Dr.mmbuddekar, Consult.kirthi, Edinwiki, Dsnelling, Visvadinu, AbdulKhaaliq2, Julienj, Jesse V., RjwilmsiBot, 4483APK, Ripchip Bot, Bookiewookie, Andrey86, Dennislees, DASHBot, EmausBot, John of Reading, Davejohnsan, Coolbloke94, Detnos, Arindra r, Dewritech, Faolin42, TheSoundAndTheFury, Arthur Davies Sikopo, Bhat sudha, Ashamerie, Fæ, Oulrij, Didym, Bamyers99, Axxonnfire, Ocean Shores, Sahimrobot, Steven-arts, Ochado, Status, Thiotimoline, Mshivaram.ie22, 28bot, ClueBot NG, Kompowiec2, J3st, CocuBot, ClaudiaHetman, Jenova20, Lord Chamberlain, the Renowned, Whitehorse212, Solusinaeternum, Frietjes, SERIEZ, Jakuzem, Widr, Altaf.attari86, Wbm1058, JoeB34, SocialRadiusOly, Griznant, Meldraft, Qx2020, TCN7JM, [illegible], Whatsnxt, Nsda, Percede, Altaïr, Mflore4d, Shaun, Softwareqa, Lekshmann, Eric Agbozo, Ulugen, Mogism, Jamesx12345, Sriharsh1234, Snippy the heavily-templated snail, Destroybotter, Pertolepe, C5st4wr6ch, Faizan, Epicgenius, Lsmll, EagleMongoose, TrOlLLin1212, PublicAmpersand, Blergleblerg, My name is not dave, Jianhui67, RichSaunders, Cornelia Gamst, JaconaFrere, Editorfun, Usarid, Monkbot, Sofia Koutsouveli, Nohorbee, Comptly, Tpprufer, Nima1353, Sarasedgewick, Effan Ganaz YK, Vedanga Kumar, Wolfram graetz, Whereisthesun, KasparBot, Mcoblenz, C a swtest, Distle and Anonymous: 688

13.2 Images

- **File:Question_book-new.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*

  Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:* Tkgd2007

13.3 Content license

- Creative Commons Attribution-Share Alike 3.0