Reflection (computer programming)

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#p-search)

*Not to be confused with*[*Reflection (computer graphics)*](https://en.wikipedia.org/wiki/Reflection_(computer_graphics))*.*

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **reflection** is the ability of a [process](https://en.wikipedia.org/wiki/Process_(computing)) to examine, [introspect](https://en.wikipedia.org/wiki/Introspection_(computer_science)), and modify its own structure and behavior.[[1]](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#cite_note-1)

**Contents**

* [1Historical background](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Historical_background)
* [2Uses](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Uses)
* [3Implementation](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Implementation)
* [4Examples](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Examples)
  + [4.1C#](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#C#)
  + [4.2Delphi](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Delphi)
  + [4.3eC](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#eC)
  + [4.4ECMAScript](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#ECMAScript)
  + [4.5Go](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Go)
  + [4.6Java](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Java)
  + [4.7Objective-C](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Objective-C)
  + [4.8Perl](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Perl)
  + [4.9PHP](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#PHP)
  + [4.10Python](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Python)
  + [4.11R](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#R)
  + [4.12Ruby](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Ruby)
  + [4.13Xojo](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Xojo)
* [5See also](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#See_also)
* [6References](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#References)
  + [6.1Citations](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Citations)
  + [6.2Sources](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Sources)
* [7Further reading](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#Further_reading)
* [8External links](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#External_links)

Historical background[[edit](https://en.wikipedia.org/w/index.php?title=Reflection_(computer_programming)&action=edit&section=1)]

The earliest computers were programmed in their native [assembly language](https://en.wikipedia.org/wiki/Assembly_language), which were inherently reflective, as these original architectures could be programmed by defining instructions as data and using [self-modifying code](https://en.wikipedia.org/wiki/Self-modifying_code). As programming moved to compiled higher-level languages such as [Algol](https://en.wikipedia.org/wiki/ALGOL), [Cobol](https://en.wikipedia.org/wiki/Cobol), and [Fortran](https://en.wikipedia.org/wiki/Fortran) (but also [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)) and [C](https://en.wikipedia.org/wiki/C_(programming_language)) and many other languages), this reflective ability largely disappeared until programming languages with reflection built into their type systems appeared.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

[Brian Cantwell Smith](https://en.wikipedia.org/wiki/Brian_Cantwell_Smith)'s 1982 doctoral dissertation[[2]](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#cite_note-2)[[3]](https://en.wikipedia.org/wiki/Reflection_(computer_programming)#cite_note-3) introduced the notion of computational reflection in procedural [programming languages](https://en.wikipedia.org/wiki/Programming_languages) and the notion of the [meta-circular interpreter](https://en.wikipedia.org/wiki/Meta-circular_interpreter) as a component of [3-Lisp](https://en.wikipedia.org/w/index.php?title=3-Lisp&action=edit&redlink=1).

Uses[[edit](https://en.wikipedia.org/w/index.php?title=Reflection_(computer_programming)&action=edit&section=2)]

Reflection helps programmers make generic software libraries to display data, process different formats of data, perform [serialization](https://en.wikipedia.org/wiki/Serialization) or deserialization of data for communication, or do bundling and unbundling of data for containers or bursts of communication.

Effective use of reflection almost always requires a plan: A design framework, encoding description, object library, a map of a database or entity relations.

Reflection makes a language more suited to network-oriented code. For example, it assists languages such as Java to operate well in networks by enabling libraries for serialization, bundling and varying data formats. Languages without reflection (e.g. [C](https://en.wikipedia.org/wiki/C_(programming_language))) have to use auxiliary compilers, e.g. for [Abstract Syntax Notation](https://en.wikipedia.org/wiki/Abstract_Syntax_Notation), to produce code for serialization and bundling.

Reflection can be used for observing and modifying program execution at runtime. A reflection-oriented program component can monitor the execution of an enclosure of code and can modify itself according to a desired goal related to that enclosure. This is typically accomplished by dynamically assigning program code at runtime.

In object-oriented programming languages such as [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), reflection allows *inspection* of classes, interfaces, fields and methods at runtime without knowing the names of the interfaces, fields, methods at compile time. It also allows *instantiation* of new objects and *invocation* of methods.

Reflection is often used as part of [software testing](https://en.wikipedia.org/wiki/Software_testing), such as for the runtime creation/instantiation of [mock objects](https://en.wikipedia.org/wiki/Mock_object).

Reflection is also a key strategy for [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming).

In some object-oriented programming languages, such as [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), reflection can be used to bypass [member accessibility](https://en.wikipedia.org/wiki/Member_accessibility) rules. For C#-properties this can be achieved by writing directly onto the (usually invisible) backing field of a non-public property. It is also possible to find non-public methods of classes and types and manually invoke them. This works for project-internal files as well as external libraries (.Net-assemblies and Java-archives).

Implementation[[edit](https://en.wikipedia.org/w/index.php?title=Reflection_(computer_programming)&action=edit&section=3)]

A language supporting reflection provides a number of features available at runtime that would otherwise be difficult to accomplish in a lower-level language. Some of these features are the abilities to:

* Discover and modify source-code constructions (such as code blocks, [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), methods, protocols, etc.) as [first-class objects](https://en.wikipedia.org/wiki/First-class_object) at runtime.
* Convert a [string](https://en.wikipedia.org/wiki/String_(computer_science)) matching the symbolic name of a class or function into a reference to or invocation of that class or function.
* Evaluate a string as if it were a source-code statement at runtime.
* Create a new interpreter for the language's bytecode to give a new meaning or purpose for a programming construct.

These features can be implemented in different ways. In [MOO](https://en.wikipedia.org/wiki/MOO_(programming_language)), reflection forms a natural part of everyday programming idiom. When verbs (methods) are called, various variables such as *verb* (the name of the verb being called) and *this* (the object on which the verb is called) are populated to give the context of the call. Security is typically managed by accessing the caller stack programmatically: Since *callers*() is a list of the methods by which the current verb was eventually called, performing tests on *callers*()[1] (the command invoked by the original user) allows the verb to protect itself against unauthorised use.

Compiled languages rely on their runtime system to provide information about the source code. A compiled [Objective-C](https://en.wikipedia.org/wiki/Objective-C) executable, for example, records the names of all methods in a block of the executable, providing a table to correspond these with the underlying methods (or selectors for these methods) compiled into the program. In a compiled language that supports runtime creation of functions, such as [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), the runtime environment must include a compiler or an interpreter.

Reflection can be implemented for languages not having built-in reflection facilities by using a [program transformation](https://en.wikipedia.org/wiki/Program_transformation) system to define automated source-code changes.