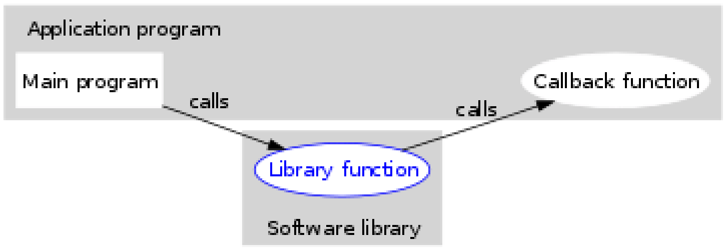
WIKIPEDIA

# Callback (computer programming)

In computer programming a **callback** is any executable code that is passed as an argument to other code, which is expected to *call back* (execute) the argument at a given time. This execution may be immediate as in a **synchronous callback**, or it might happen at a later time as in an **asynchronous callback**. In all cases, the intention is to specify a function or subroutine as an entity that is, depending on the language, more or less similar to a variable.



A callback is often back on the level of the original caller

Programming languages support callbacks in different ways, often implementing them with subroutines, lambda expressions, blocks, or function pointers.

## Contents

1 Design
2 Implementation
3 Use
3.1 C
3.2 C#
3.3 JavaScript
3.4 Lua
3.5 Python
4 See also
5 References
6 External links

## Design

There are two types of callbacks, differing in how they control data flow at runtime: *blocking callbacks* (also known as *synchronous callbacks* or just *callbacks*) and *deferred callbacks* (also known as *asynchronous callbacks*). While blocking callbacks are invoked before a function returns (in the C example below, which illustrates a blocking callback, it is function `main`), deferred callbacks may be invoked after a function returns. Deferred callbacks are often used in the context of I/O operations or event handling, and are called by interrupts or by a different thread in case of multiple threads. Due to their nature, blocking callbacks can work without interrupts or multiple threads, meaning that blocking callbacks are not commonly used for synchronization or delegating work to another thread.

Callbacks are used to program applications in windowing systems. In this case, the application supplies (a reference to) a specific custom callback function for the operating system to call, which then calls this application-specific function in response to events like mouse clicks or key presses. A major concern here is the management of privilege and security: whilst the function is called from the operating system, it should not run with the same privilege as the system. A solution to this problem is using rings of protection.

## Implementation

The form of a callback varies among programming languages:

- In assembly, C, C++, Pascal, Modula2 and similar languages, a machine-level pointer to a function may be passed as an argument to another (internal or external) function. This is supported by most compilers and provides the advantage of using different languages together without special wrapper libraries or classes. One example may be the Windows API that is directly (more or less) accessible by many different languages, compilers and assemblers.
- C++ allows objects to provide their own implementation of the function call operation. The Standard Template Library accepts these objects (called *functors*), as well as function pointers, as parameters to various polymorphic algorithms.
- Many interpreted languages, such as JavaScript, Lua, Python, Perl[1][2] and PHP, simply allow a function object to be passed through.
- CLI languages such as C# and VB.NET provide a type-safe encapsulating reference, a "delegate", to define well-typed function pointers. These can be used as callbacks.
- Events and event handlers, as used in .NET languages, provide generalized syntax for callbacks.
- Functional languages generally support first-class functions, which can be passed as callbacks to other functions, stored as data or returned from functions.
- Some languages, such as Algol 68, Perl, Python, Ruby, Smalltalk, C++11 and later, newer versions of C# and VB.NET as well as most functional languages, allow unnamed blocks of code (lambda expressions) to be supplied instead of references to functions defined elsewhere.
- In some languages, e.g. Scheme, ML, JavaScript, Perl, Smalltalk, PHP (since 5.3.0),[3] C++11 and later, Java (since 8),[4] and many others, such functions can be closures, i.e. they can access and modify variables locally defined in the context in which the function was defined. Note that Java cannot, however, modify the local variables in the enclosing scope.
- In object-oriented programming languages without function-valued arguments, such as in Java before its 8 version, callbacks can be simulated by passing an instance of an abstract class or interface, of which the receiver will call one or more methods, while the calling end provides a concrete implementation. Such objects are effectively a bundle of callbacks, plus the data they need to manipulate. They are useful in implementing various design patterns such as Visitor, Observer, and Strategy.

## Use

### C

Callbacks have a wide variety of uses, for example in error signaling: a Unix program might not want to terminate immediately when it receives SIGTERM, so to make sure that its termination is handled properly, it would register the cleanup function as a callback. Callbacks may also be used to control whether a function acts or not: Xlib allows custom predicates to be specified to determine whether a program wishes to handle an event.

The following C code demonstrates the use of callbacks to display two numbers.

```c
#include <stdio.h>
#include <stdlib.h>

/* The calling function takes a single callback as a parameter. */
void PrintTwoNumbers(int (*numberSource)(void)) {
    int val1= numberSource();
    int val2= numberSource();
    printf("%d and %d\n", val1, val2);
}

/* A possible callback */
int overNineThousand(void) {
    return (rand()%1000) + 9001;
}

/* Another possible callback. */
int meaningOfLife(void) {
    return 42;
}

/* Here we call PrintTwoNumbers() with three different callbacks. */
int main(void) {
    PrintTwoNumbers(&rand);
    PrintTwoNumbers(&overNineThousand);
    PrintTwoNumbers(&meaningOfLife);
    return 0;
}
```

This should provide output similar to:

```
125185 and 89187225
 9084 and 9441
 42 and 42
```

Note how this is different from simply passing the output of the callback function to the calling function, PrintTwoNumbers() - rather than printing the same value twice, the PrintTwoNumbers calls the callback as many times as it requires. This is one of the two main advantages of callbacks.

The other advantage is that the calling function can pass whatever parameters it wishes to the called functions (not shown in the above example). This allows correct information hiding: the code that passes a callback to a calling function does not need to know the parameter values that will be passed to the function. If it only passed the return value, then the parameters would need to be exposed publicly.

Another example:

```
/*
 * This is a simple C program to demonstrate the usage of callbacks
 * The callback function is in the same file as the calling code.
 * The callback function can later be put into external library like
 * e.g. a shared object to increase flexibility.
 *
 */

#include <stdio.h>
#include <string.h>

typedef struct _MyMsg {
    int appId;
    char msgbody[32];
} MyMsg;

void myfunc(MyMsg *msg)
{
    if (strlen(msg->msgbody) > 0 )
        printf("App Id = %d \nMsg = %s \n",msg->appId, msg->msgbody);
    else
        printf("App Id = %d \nMsg = No Msg\n",msg->appId);
}

/*
 * Prototype declaration
 */
void (*callback)(MyMsg *);

int main(void)
{
    MyMsg msg1;
    msg1.appId = 100;
    strcpy(msg1.msgbody, "This is a test\n");

    /*
     * Assign the address of the function "myfunc" to the function
     * pointer "callback" (may be also written as "callback = &myfunc;")
     */
    callback = myfunc;

    /*
     * Call the function (may be also written as "(*callback)(&msg1);")
     */
    callback(&msg1);

    return 0;
}
```

The output after compilation:

```
$ gcc cbtest.c
$ ./a.out
```

```
App Id = 100
Msg = This is a test
```

This information hiding means that callbacks can be used when communicating between processes or threads, or through serialised communications and tabular data.

## C#

A simple callback in C#:

```
public class Class1
{
    static void Main(string[] args)
    {
        Class2 c2 = new Class2();

        /*
        * Calling method on Class2 with callback method as parameter
        */
        c2.Method(CallBackMet);
    }

    /*
    * The callback method. This method prints the string sent in the callback
    */
    static void CallBackMet(string str)
    {
        Console.WriteLine("Callback was: " + str);
    }
}

public class Class2
{
    /*
    * The method that calls back to the caller. Takes an action (method) as parameter
    */
    public void Method(Action<string> callback)
    {
        /*
        * Calls back to method CallBackMet in Class1 with the message specified
        */
        callback("The message to send back");
    }
}
```

## JavaScript

Callbacks are used in the implementation of languages such as JavaScript, including support of JavaScript functions as callbacks through js-ctypes[5] and in components such as addEventListener.[6] However, a naive example of a callback can be written without any complex code:

```
function someAction(x, y, someCallback) {
    return someCallback(x, y);
}

function calcProduct(x, y) {
    return x * y;
}

function calcSum(x, y) {
    return x + y;
}
// alerts 75, the product of 5 and 15
alert(someAction(5, 15, calcProduct));
// alerts 20, the sum of 5 and 15
alert(someAction(5, 15, calcSum));
```

First a function `someAction` is defined with a parameter intended for callback: `someCallback`. Then a function that can be used as a callback to `someAction` is defined, `calcProduct`. Other functions may be used for `someCallback`, like `calcSum`. In this example, `someAction()` is invoked twice, once with `calcProduct` as a callback and once with `calcSum`. The functions return the product and sum, respectively, and then the alert will display them to the screen.

In this primitive example, the use of a callback is primarily a demonstration of principle. One could simply call the callbacks as regular functions, `calcProduct(x, y)`. Callbacks are generally useful when the function needs to perform actions before the callback is executed, or when the function does not (or cannot) have meaningful return values to act on, as is the case for Asynchronous JavaScript (based on timers) or XMLHttpRequest requests. Useful examples can be found in JavaScript libraries such as jQuery where the .each() method iterates over an array-like object, the first argument being a callback that is performed on each iteration.

## Lua

A color tweening example using the ROBLOX engine that takes an optional .done callback:

```
wait(1)
local DT = wait()

function tween_color(object, finish_color, fade_time)
  local step_r = finish_color.r - object.BackgroundColor3.r
  local step_g = finish_color.g - object.BackgroundColor3.g
  local step_b = finish_color.b - object.BackgroundColor3.b
  local total_steps = 1/(DT*(1/fade_time))
  local completed;
  coroutine.wrap(function()
    for i = 0, 1, DT*(1/fade_time) do
      object.BackgroundColor3 = Color3.new (
        object.BackgroundColor3.r + (step_r/total_steps),
        object.BackgroundColor3.g + (step_g/total_steps),
        object.BackgroundColor3.b + (step_b/total_steps)
      )
      wait()
    end
    if completed then
      completed()
    end
  end)()
  return {
    done = function(callback)
      completed = callback
    end
  }
end

tween_color(some_object, Color3.new(1, 0, 0), 1).done(function()
  print "Color tweening finished!"
end)
```

## Python

A classic use of callbacks in Python (and other languages) is to assign events to UI elements.

Here is a very trivial example of the use of a callback in Python. First define two functions, the callback and the calling code, then pass the callback function into the calling code.

```
>>> def my_square(val):
...     """ the callback """
...     return val ** 2
...
>>> def caller(val, func):
...     return func(val)
...
>>> for i in range(5):
...     j = caller(i, my_square)
...     print("{0} ** 2 = {1}".format(i, j))
...
0 ** 2 = 0
1 ** 2 = 1
2 ** 2 = 4
3 ** 2 = 9
4 ** 2 = 16
```

## See also

- Command pattern
- Continuation-passing style
- Event loop
- Event-driven programming
- Implicit invocation
- Inversion of control
- libsigc++, a callback library for C++
- Signals and slots
- User exit

## References

1. "Perl Cookbook - 11.4. Taking References to Functions" (http://www.unix.org.ua/orelly/perl/cookbook/ch11_05.htm). Retrieved 2008-03-03.
2. "Advanced Perl Programming - 4.2 Using Subroutine References" (http://www.unix.org.ua/orelly/perl/advprog/ch04_02.htm). Retrieved 2008-03-03.
3. "PHP Language Reference - Anonymous functions" (https://secure.php.net/manual/en/functions.anonymous.php). Retrieved 2011-06-08.
4. "What's New in JDK 8" (http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html). *oracle.com*.
5. "Callbacks" (https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes/js-ctypes_reference/Callbacks). Mozilla Developer Network. Retrieved 13 December 2012.
6. "Creating Javascript Callbacks in Components" (https://developer.mozilla.org/en-US/docs/Creating_JavaScript_callbacks_in_components#JavaScript_functions_as_callbacks). Mozilla Developer Network. Retrieved 13 December 2012.

## External links

- Basic Instincts: Implementing Callback Notifications Using Delegates
- Implement callback routines in Java
- Implement Script Callback Framework in ASP.NET
- Interfacing C++ member functions with C libraries (archived from the original on July 6, 2011)
- Style Case Study #2: Generic Callbacks

Retrieved from "https://en.wikipedia.org/w/index.php?title=Callback_(computer_programming)&oldid=809489307"