A delegate is a type that safely encapsulates a method, similar to a function pointer in C and C++. Unlike C function pointers, delegates are object-oriented, type safe, and secure. The type of a delegate is defined by the name of the delegate. The following example declares a delegate named Del that can encapsulate a method that takes a string as an argument and returns [void](https://msdn.microsoft.com/en-us/library/yah0tteb.aspx):

C#

public delegate void Del(string message);

A delegate object is normally constructed by providing the name of the method the delegate will wrap, or with an anonymous Method. Once a delegate is instantiated, a method call made to the delegate will be passed by the delegate to that method. The parameters passed to the delegate by the caller are passed to the method, and the return value, if any, from the method is returned to the caller by the delegate. This is known as invoking the delegate. An instantiated delegate can be invoked as if it were the wrapped method itself. For example:

C#

// Create a method for a delegate.

public static void DelegateMethod(string message)

{

System.Console.WriteLine(message);

}

C#

// Instantiate the delegate.

Del handler = DelegateMethod;

// Call the delegate.

handler("Hello World");

Delegate types are derived from the Delegate class in the .NET Framework. Delegate types are sealed—they cannot be derived from— and it is not possible to derive custom classes from Delegate. Because the instantiated delegate is an object, it can be passed as a parameter, or assigned to a property. This allows a method to accept a delegate as a parameter, and call the delegate at some later time

This is especially powerful since a delegated method can use any number of parameters.