Delegates (C# Programming Guide)

Article • 09/29/2022

A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.

Delegates are used to pass methods as arguments to other methods. Event handlers are nothing more than methods that are invoked through delegates. You create a custom method, and a class such as a windows control can call your method when a certain event occurs. The following example shows a delegate declaration:

C#

```
public delegate int PerformCalculation(int x, int y);
```

Any method from any accessible class or struct that matches the delegate type can be assigned to the delegate. The method can be either static or an instance method. This flexibility means you can programmatically change method calls, or plug new code into existing classes.

> **Note**
>
> In the context of method overloading, the signature of a method does not include the return value. But in the context of delegates, the signature does include the return value. In other words, a method must have the same return type as the delegate.

This ability to refer to a method as a parameter makes delegates ideal for defining callback methods. You can write a method that compares two objects in your application. That method can be used in a delegate for a sort algorithm. Because the comparison code is separate from the library, the sort method can be more general.

Function pointers support similar scenarios, where you need more control over the calling convention. The code associated with a delegate is invoked using a virtual method added to a delegate type. Using function pointers, you can specify different conventions.

Delegates Overview

Delegates have the following properties:

- Delegates are similar to C++ function pointers, but delegates are fully object-oriented, and unlike C++ pointers to member functions, delegates encapsulate both an object instance and a method.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- Methods don't have to match the delegate type exactly. For more information, see Using Variance in Delegates.
- Lambda expressions are a more concise way of writing inline code blocks. Lambda expressions (in certain contexts) are compiled to delegate types. For more information about lambda expressions, see Lambda expressions.

In This Section

- Using Delegates
- When to Use Delegates Instead of Interfaces (C# Programming Guide)
- Delegates with Named vs. Anonymous Methods
- Using Variance in Delegates
- How to combine delegates (Multicast Delegates)
- How to declare, instantiate, and use a delegate

C# Language Specification

For more information, see Delegates in the C# Language Specification. The language specification is the definitive source for C# syntax and usage.

See also

- Delegate
- Events

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback