virtual (C# Reference)

<http://msdn.microsoft.com/en-us/library/9fkccyh4(v=vs.100).aspx>

**Visual Studio 2010**

[Other Versions](javascript:;)

12 out of 16 rated this helpful - [Rate this topic](http://msdn.microsoft.com/en-us/library/9fkccyh4(v=vs.100).aspx#feedback)

The **virtual** keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class. For example, this method can be overridden by any class that inherits it:

public virtual double Area()

{

return x \* y;

}

The implementation of a virtual member can be changed by an [overriding member](http://msdn.microsoft.com/en-us/library/ebca9ah3(v=vs.100).aspx) in a derived class. For more information about how to use the **virtual** keyword, see [Versioning with the Override and New Keywords (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/6fawty39(v=vs.100).aspx) and [Knowing When to Use Override and New Keywords (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/ms173153(v=vs.100).aspx).

[Remarks](javascript:void(0))

When a virtual method is invoked, the run-time type of the object is checked for an overriding member. The overriding member in the most derived class is called, which might be the original member, if no derived class has overridden the member.

By default, methods are non-virtual. You cannot override a non-virtual method.

You cannot use the **virtual** modifier with the **static**, **abstract, private**, or **override** modifiers. The following example shows a virtual property:

C#

class MyBaseClass

{

// virtual auto-implemented property. Overrides can only

// provide specialized behavior if they implement get and set accessors.

public virtual string Name { get; set; }

// ordinary virtual property with backing field

private int num;

public virtual int Number

{

get { return num; }

set { num = value; }

}

}

class MyDerivedClass : MyBaseClass

{

private string name;

// Override auto-implemented property with ordinary property

// to provide specialized accessor behavior.

public override string Name

{

get

{

return name;

}

set

{

if (value != String.Empty)

{

name = value;

}

else

{

name = "Unknown";

}

}

}

}

Virtual properties behave like abstract methods, except for the differences in declaration and invocation syntax.

* It is an error to use the **virtual** modifier on a static property.
* A virtual inherited property can be overridden in a derived class by including a property declaration that uses the **override** modifier.

[Example](javascript:void(0))

In this example, the Shape class contains the two coordinates x, y, and the Area() virtual method. Different shape classes such as Circle, Cylinder, and Sphereinherit the Shape class, and the surface area is calculated for each figure. Each derived class has it own override implementation of Area().

Notice that the inherited classes Circle, Sphere, and Cylinder all use constructors that initialize the base class, as shown in the following declaration.

public Cylinder(double r, double h): base(r, h) {}

The following program calculates and displays the appropriate area for each figure by invoking the appropriate implementation of the Area() method, according to the object that is associated with the method.

C#

class TestClass

{

public class Shape

{

public const double PI = Math.PI;

protected double x, y;

public Shape()

{

}

public Shape(double x, double y)

{

this.x = x;

this.y = y;

}

public virtual double Area()

{

return x \* y;

}

}

public class Circle : Shape

{

public Circle(double r) : base(r, 0)

{

}

public override double Area()

{

return PI \* x \* x;

}

}

class Sphere : Shape

{

public Sphere(double r) : base(r, 0)

{

}

public override double Area()

{

return 4 \* PI \* x \* x;

}

}

class Cylinder : Shape

{

public Cylinder(double r, double h) : base(r, h)

{

}

public override double Area()

{

return 2 \* PI \* x \* x + 2 \* PI \* x \* y;

}

}

static void Main()

{

double r = 3.0, h = 5.0;

Shape c = new Circle(r);

Shape s = new Sphere(r);

Shape l = new Cylinder(r, h);

// Display results:

Console.WriteLine("Area of Circle = {0:F2}", c.Area());

Console.WriteLine("Area of Sphere = {0:F2}", s.Area());

Console.WriteLine("Area of Cylinder = {0:F2}", l.Area());

}

}

/\*

Output:

Area of Circle = 28.27

Area of Sphere = 113.10

Area of Cylinder = 150.80

\*/

[C# Language Specification](javascript:void(0))

For more information, see the [C# Language Specification](http://msdn.microsoft.com/en-us/library/ms228593(v=vs.100).aspx). The language specification is the definitive source for C# syntax and usage.

[See Also](javascript:void(0))

Reference

[Modifiers (C# Reference)](http://msdn.microsoft.com/en-us/library/6tcf2h8w(v=vs.100).aspx)

[C# Keywords](http://msdn.microsoft.com/en-us/library/x53a06bb(v=vs.100).aspx)

[Polymorphism (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/ms173152(v=vs.100).aspx)

[abstract (C# Reference)](http://msdn.microsoft.com/en-us/library/sf985hc5(v=vs.100).aspx)

[override (C# Reference)](http://msdn.microsoft.com/en-us/library/ebca9ah3(v=vs.100).aspx)

[new (C# Reference)](http://msdn.microsoft.com/en-us/library/51y09td4(v=vs.100).aspx)

Concepts

[C# Programming Guide](http://msdn.microsoft.com/en-us/library/67ef8sbd(v=vs.100).aspx)

Other Resources

[C# Reference](http://msdn.microsoft.com/en-us/library/618ayhy6(v=vs.100).aspx)