Visual Studio 2010 - Visual C#

**out (Generic Modifier) (C# Reference)**

For generic type parameters, the **out** keyword specifies that the type parameter is covariant. You can use the **out** keyword in generic interfaces and delegates.

Covariance enables you to use a more derived type than that specified by the generic parameter. This allows for implicit conversion of classes that implement variant interfaces and implicit conversion of delegate types. Covariance and contravariance are supported for reference types, but they are not supported for value types.

An interface that has a covariant type parameter enables its methods to return more derived types than those specified by the type parameter. For example, because in .NET Framework 4, in [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx), type T is covariant, you can assign an object of the IEnumerabe(Of String) type to an object of the IEnumerable(Of Object) type without using any special conversion methods.

A covariant delegate can be assigned another delegate of the same type, but with a more derived generic type parameter.

For more information, see [Covariance and Contravariance (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/ee207183.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

The following example shows how to declare, extend, and implement a covariant generic interface. It also shows how to use implicit conversion for classes that implement a covariant interface.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl00_code');" \o "Copy Code)

// Covariant interface.

interface ICovariant<out R> { }

// Extending covariant interface.

interface IExtCovariant<out R> : ICovariant<R> { }

// Implementing covariant interface.

class Sample<R> : ICovariant<R> { }

class Program

{

static void Test()

{

ICovariant<Object> iobj = new Sample<Object>();

ICovariant<String> istr = new Sample<String>();

// You can assign istr to iobj because

// the ICovariant interface is covariant.

iobj = istr;

}

}

In a generic interface, a type parameter can be declared covariant if it satisfies the following conditions:

* The type parameter is used only as a return type of interface methods and not used as a type of method arguments.

|  |
| --- |
| **Description: NoteNote** |
| There is one exception to this rule. If in a covariant interface you have a contravariant generic delegate as a method parameter, you can use the covariant type as a generic type parameter for this delegate. For more information about covariant and contravariant generic delegates, see [Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233060.aspx) and [Using Variance for Func and Action Generic Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd465122.aspx). |

* The type parameter is not used as a generic constraint for the interface methods.

The following example shows how to declare, instantiate, and invoke a covariant generic delegate. It also shows how to implicitly convert delegate types.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl04_code');" \o "Copy Code)

// Covariant delegate.

public delegate R DCovariant<out R>();

// Methods that match the delegate signature.

public static Control SampleControl()

{ return new Control(); }

public static Button SampleButton()

{ return new Button(); }

public void Test()

{

// Instantiate the delegates with the methods.

DCovariant<Control> dControl = SampleControl;

DCovariant<Button> dButton = SampleButton;

// You can assign dButton to dControl

// because the DCovariant delegate is covariant.

dControl = dButton;

// Invoke the delegate.

dControl();

}

In a generic delegate, a type can be declared covariant if it is used only as a method return type and not used for method arguments.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifC# Language Specification

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