Visual Studio 2010 - Visual C#

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

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

Contravariance enables you to use a less 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 in generic type parameters are supported for reference types, but they are not supported for value types.

A type can be declared contravariant in a generic interface or delegate if it is used only as a type of method arguments and not used as a method return type. **Ref** and **out** parameters cannot be variant.

An interface that has a contravariant type parameter allows its methods to accept arguments of less derived types than those specified by the interface type parameter. For example, because in .NET Framework 4, in the [IComparer<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/8ehhxeaf.aspx) interface, type T is contravariant, you can assign an object of the IComparer(Of Person) type to an object of the IComparer(Of Employee) type without using any special conversion methods if Employee inherits Person.

A contravariant delegate can be assigned another delegate of the same type, but with a less 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 contravariant generic interface. It also shows how you can use implicit conversion for classes that implement this interface.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_ctl00_ctl00_code');" \o "Copy Code)

// Contravariant interface.

interface IContravariant<in A> { }

// Extending contravariant interface.

interface IExtContravariant<in A> : IContravariant<A> { }

// Implementing contravariant interface.

class Sample<A> : IContravariant<A> { }

class Program

{

static void Test()

{

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

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

// You can assign iobj to istr because

// the IContravariant interface is contravariant.

istr = iobj;

}

}

The following example shows how to declare, instantiate, and invoke a contravariant generic delegate. It also shows how you can implicitly convert a delegate type.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_ctl00_ctl01_code');" \o "Copy Code)

// Contravariant delegate.

public delegate void DContravariant<in A>(A argument);

// Methods that match the delegate signature.

public static void SampleControl(Control control)

{ }

public static void SampleButton(Button button)

{ }

public void Test()

{

// Instantiating the delegates with the methods.

DContravariant<Control> dControl = SampleControl;

DContravariant<Button> dButton = SampleButton;

// You can assign dControl to dButton

// because the DContravariant delegate is contravariant.

dButton = dControl;

// Invoke the delegate.

dButton(new Button());

}

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.