Visual Studio 2010

**Covariance and Contravariance (C# and Visual Basic)**

In C# and Visual Basic, covariance and contravariance enable implicit reference conversion for array types, delegate types, and generic type arguments. Covariance preserves assignment compatibility and contravariance reverses it.

The following code demonstrates the difference between assignment compatibility, covariance, and contravariance.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

' Assignment compatibility.

Dim str As String = "test"

' An object of a more derived type is assigned to an object of a less derived type.

Dim obj As Object = str

' Covariance.

Dim strings As IEnumerable(Of String) = New List(Of String)()

' An object that is instantiated with a more derived type argument

' is assigned to an object instantiated with a less derived type argument.

' Assignment compatibility is preserved.

Dim objects As IEnumerable(Of Object) = strings

' Contravariance.

' Assume that there is the following method in the class:

' Shared Sub SetObject(ByVal o As Object)

' End Sub

Dim actObject As Action(Of Object) = AddressOf SetObject

' An object that is instantiated with a less derived type argument

' is assigned to an object instantiated with a more derived type argument.

' Assignment compatibility is reversed.

Dim actString As Action(Of String) = actObject

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl02_code');" \o "Copy Code)

// Assignment compatibility.

string str = "test";

// An object of a more derived type is assigned to an object of a less derived type.

object obj = str;

// Covariance.

IEnumerable<string> strings = new List<string>();

// An object that is instantiated with a more derived type argument

// is assigned to an object instantiated with a less derived type argument.

// Assignment compatibility is preserved.

IEnumerable<object> objects = strings;

// Contravariance.

// Assume that the following method is in the class:

// static void SetObject(object o) { }

Action<object> actObject = SetObject;

// An object that is instantiated with a less derived type argument

// is assigned to an object instantiated with a more derived type argument.

// Assignment compatibility is reversed.

Action<string> actString = actObject;

Covariance for arrays enables implicit conversion of an array of a more derived type to an array of a less derived type. But this operations is not type safe, as shown in the following code example.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl03_code');" \o "Copy Code)

Dim array() As Object = New String(10) {}

' The following statement produces a run-time exception.

' array(0) = 10

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl04_code');" \o "Copy Code)

object[] array = new String[10];

// The following statement produces a run-time exception.

// array[0] = 10;

Covariance and contravariance support for method groups allows for matching method signatures with delegate types. This enables you to assign to delegates not only methods that have matching signatures, but also methods that return more derived types (covariance) or that accept parameters that have less derived types (contravariance) than that specified by the delegate type. For more information, see [Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233060.aspx) and [Using Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/ms173174.aspx).

The following code example shows covariance and contravariance support for method groups.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl07_code');" \o "Copy Code)

Shared Function GetObject() As Object

Return Nothing

End Function

Shared Sub SetObject(ByVal obj As Object)

End Sub

Shared Function GetString() As String

Return ""

End Function

Shared Sub SetString(ByVal str As String)

End Sub

Shared Sub Test()

' Covariance. A delegate specifies a return type as object,

' but you can assign a method that returns a string.

Dim del As Func(Of Object) = AddressOf GetString

' Contravariance. A delegate specifies a parameter type as string,

' but you can assign a method that takes an object.

Dim del2 As Action(Of String) = AddressOf SetObject

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl08_code');" \o "Copy Code)

static object GetObject() { return null; }

static void SetObject(object obj) { }

static string GetString() { return ""; }

static void SetString(string str) { }

static void Test()

{

// Covariance. A delegate specifies a return type as object,

// but you can assign a method that returns a string.

Func<object> del = GetString;

// Contravariance. A delegate specifies a parameter type as string,

// but you can assign a method that takes an object.

Action<string> del2 = SetObject;

}

In .NET Framework 4 and Visual Studio 2010, both C# and Visual Basic support covariance and contravariance in generic interfaces and delegates and allow for implicit conversion of generic type parameters. For more information, see [Variance in Generic Interfaces (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233059.aspx) and [Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233060.aspx).

The following code example shows implicit reference conversion for generic interfaces.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl11_code');" \o "Copy Code)

Dim strings As IEnumerable(Of String) = New List(Of String)

Dim objects As IEnumerable(Of Object) = strings

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_code');" \o "Copy Code)

IEnumerable<String> strings = new List<String>();

IEnumerable<Object> objects = strings;

A generic interface or delegate is called *variant* if its generic parameters are declared covariant or contravariant. Both C# and Visual Basic enable you to create your own variant interfaces and delegates. For more information, see [Creating Variant Generic Interfaces (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd997386.aspx) and [Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233060.aspx).

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

|  |  |
| --- | --- |
| **Title** | **Description** |
| [Variance in Generic Interfaces (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233059.aspx) | Discusses covariance and contravariance in generic interfaces and provides a list of variant generic interfaces in the .NET Framework. |
| [Creating Variant Generic Interfaces (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd997386.aspx) | Shows how to create custom variant interfaces. |
| [Using Variance in Interfaces for Generic Collections (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd465120.aspx) | Shows how covariance and contravariance support in the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) and [IComparable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/4d7sx9hd.aspx) interfaces can help you reuse code. |
| [Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233060.aspx) | Discusses covariance and contravariance in generic and non-generic delegates and provides a list of variant generic delegates in the .NET Framework. |
| [Using Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/ms173174.aspx) | Shows how to use covariance and contravariance support in non-generic delegates to match method signatures with delegate types. |
| [Using Variance for Func and Action Generic Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd465122.aspx) | Shows how covariance and contravariance support in the **Func** and **Action** delegates can help you reuse code. |