Visual Studio 2010

**Variance in Delegates (C# and Visual Basic)**

.NET Framework 3.5 and Visual Studio 2008 introduced variance support for matching method signatures with delegate types in all delegates in C# and Visual Basic. This means that you can 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. This includes both generic and non-generic delegates.

For example, consider the following code, which has two classes and two delegates: generic and non-generic.

Visual Basic

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

Public Class First

End Class

Public Class Second

Inherits First

End Class

Public Delegate Function SampleDelegate(ByVal a As Second) As First

Public Delegate Function SampleGenericDelegate(Of A, R)(ByVal a As A) As R

C#

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

public class First { }

public class Second : First { }

public delegate First SampleDelegate(Second a);

public delegate R SampleGenericDelegate<A, R>(A a);

When you create delegates of the SampleDelegate or SampleGenericDelegate<A, R> (SampleDelegate(Of A, R) in Visual Basic) types, you can assign any one of the following methods to those delegates.

Visual Basic

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

' Matching signature.

Public Shared Function ASecondRFirst(

ByVal second As Second) As First

Return New First()

End Function

' The return type is more derived.

Public Shared Function ASecondRSecond(

ByVal second As Second) As Second

Return New Second()

End Function

' The argument type is less derived.

Public Shared Function AFirstRFirst(

ByVal first As First) As First

Return New First()

End Function

' The return type is more derived

' and the argument type is less derived.

Public Shared Function AFirstRSecond(

ByVal first As First) As Second

Return New Second()

End Function

C#

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

// Matching signature.

public static First ASecondRFirst(Second first)

{ return new First(); }

// The return type is more derived.

public static Second ASecondRSecond(Second second)

{ return new Second(); }

// The argument type is less derived.

public static First AFirstRFirst(First first)

{ return new First(); }

// The return type is more derived

// and the argument type is less derived.

public static Second AFirstRSecond(First first)

{ return new Second(); }

The following code example illustrates the implicit conversion between the method signature and the delegate type.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl05_code');" \o "Copy Code)

' Assigning a method with a matching signature

' to a non-generic delegate. No conversion is necessary.

Dim dNonGeneric As SampleDelegate = AddressOf ASecondRFirst

' Assigning a method with a more derived return type

' and less derived argument type to a non-generic delegate.

' The implicit conversion is used.

Dim dNonGenericConversion As SampleDelegate = AddressOf AFirstRSecond

' Assigning a method with a matching signature to a generic delegate.

' No conversion is necessary.

Dim dGeneric As SampleGenericDelegate(Of Second, First) = AddressOf ASecondRFirst

' Assigning a method with a more derived return type

' and less derived argument type to a generic delegate.

' The implicit conversion is used.

Dim dGenericConversion As SampleGenericDelegate(Of Second, First) = AddressOf AFirstRSecond

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl06_code');" \o "Copy Code)

// Assigning a method with a matching signature

// to a non-generic delegate. No conversion is necessary.

SampleDelegate dNonGeneric = ASecondRFirst;

// Assigning a method with a more derived return type

// and less derived argument type to a non-generic delegate.

// The implicit conversion is used.

SampleDelegate dNonGenericConversion = AFirstRSecond;

// Assigning a method with a matching signature to a generic delegate.

// No conversion is necessary.

SampleGenericDelegate<Second, First> dGeneric = ASecondRFirst;

// Assigning a method with a more derived return type

// and less derived argument type to a generic delegate.

// The implicit conversion is used.

SampleGenericDelegate<Second, First> dGenericConversion = AFirstRSecond;

For more examples, see [Using Variance in Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/ms173174.aspx) and [Using Variance for Func and Action Generic Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd465122.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifVariance in Generic Type Parameters

In .NET Framework 4, you can enable implicit conversion between delegates, so that generic delegates that have different types specified by generic type parameters can be assigned to each other, if the types are inherited from each other as required by variance.

To enable implicit conversion, you must explicitly declare generic parameters in a delegate as covariant or contravariant by using the **in** or **out** keyword.

The following code example shows how you can create a delegate that has a covariant generic type parameter.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl00_code');" \o "Copy Code)

' Type T is declared covariant by using the out keyword.

Public Delegate Function SampleGenericDelegate(Of Out T)() As T

Sub Test()

Dim dString As SampleGenericDelegate(Of String) = Function() " "

' You can assign delegates to each other,

' because the type T is declared covariant.

Dim dObject As SampleGenericDelegate(Of Object) = dString

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl01_code');" \o "Copy Code)

// Type T is declared covariant by using the out keyword.

public delegate T SampleGenericDelegate <out T>();

public static void Test()

{

SampleGenericDelegate <String> dString = () => " ";

// You can assign delegates to each other,

// because the type T is declared covariant.

SampleGenericDelegate <Object> dObject = dString;

}

If you use only variance support to match method signatures with delegate types and do not use the **in** and **out** keywords, you may find that sometimes you can instantiate delegates with identical lambda expressions or methods, but you cannot assign one delegate to another.

In the following code example, SampleGenericDelegate<String> cannot be explicitly converted to SampleGenericDelegate<Object> (SampleGenericDelegate(Of String) to SampleGenericDelegate(Of Object) in Visual Basic), although String inherits Object. You can fix this problem by marking the generic parameter T with the **out** keyword.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl02_code');" \o "Copy Code)

Public Delegate Function SampleGenericDelegate(Of T)() As T

Sub Test()

Dim dString As SampleGenericDelegate(Of String) = Function() " "

' You can assign the dObject delegate

' to the same lambda expression as dString delegate

' because of the variance support for

' matching method signatures with delegate types.

Dim dObject As SampleGenericDelegate(Of Object) = Function() " "

' The following statement generates a compiler error

' because the generic type T is not marked as covariant.

' Dim dObject As SampleGenericDelegate(Of Object) = dString

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl03_code');" \o "Copy Code)

public delegate T SampleGenericDelegate<T>();

public static void Test()

{

SampleGenericDelegate<String> dString = () => " ";

// You can assign the dObject delegate

// to the same lambda expression as dString delegate

// because of the variance support for

// matching method signatures with delegate types.

SampleGenericDelegate<Object> dObject = () => " ";

// The following statement generates a compiler error

// because the generic type T is not marked as covariant.

// SampleGenericDelegate <Object> dObject = dString;

}

**Generic Delegates That Have Variant Type Parameters in the .NET Framework**

.NET Framework 4 introduces variance support for generic type parameters in several existing generic delegates:

* **Action** delegates from the [System](http://msdn.microsoft.com/en-us/library/system.aspx) namespace, for example, [Action<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/018hxwa8.aspx) and [Action<(Of <(T1, T2>)>)](http://msdn.microsoft.com/en-us/library/bb549311.aspx)
* **Func** delegates from the [System](http://msdn.microsoft.com/en-us/library/system.aspx) namespace, for example, [Func<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb534960.aspx) and [Func<(Of <(T, TResult>)>)](http://msdn.microsoft.com/en-us/library/bb549151.aspx)
* The [Predicate<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/bfcke1bz.aspx) delegate
* The [Comparison<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/tfakywbh.aspx) delegate
* The [Converter<(Of <(TInput, TOutput>)>)](http://msdn.microsoft.com/en-us/library/kt456a2y.aspx) delegate

For more information and examples, see [Using Variance for Func and Action Generic Delegates (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd465122.aspx).

**Declaring Variant Type Parameters in Generic Delegates**

If a generic delegate has covariant or contravariant generic type parameters, it can be referred to as a *variant generic delegate*.

You can declare a generic type parameter covariant in a generic delegate by using the **out** keyword. The covariant type can be used only as a method return type and not as a type of method arguments. The following code example shows how to declare a covariant generic delegate.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl14_code');" \o "Copy Code)

Public Delegate Function DCovariant(Of Out R)() As R

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl15_code');" \o "Copy Code)

public delegate R DCovariant<out R>();

You can declare a generic type parameter contravariant in a generic delegate by using the **in** keyword. The contravariant type can be used only as a type of method arguments and not as a method return type. The following code example shows how to declare a contravariant generic delegate.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl16_code');" \o "Copy Code)

Public Delegate Sub DContravariant(Of In A)(ByVal a As A)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl17_code');" \o "Copy Code)

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

|  |
| --- |
| **Description: Important noteImportant** |
| **ByRef** parameters in Visual Basic and **ref** and **out** parameters in C# and cannot be marked as variant. |

It is also possible to support both variance and covariance in the same delegate, but for different type parameters. This is shown in the following example.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl19_code');" \o "Copy Code)

Public Delegate Function DVariant(Of In A, Out R)(ByVal a As A) As R

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl20_code');" \o "Copy Code)

public delegate R DVariant<in A, out R>(A a);

**Instantiating and Invoking Variant Generic Delegates**

You can instantiate and invoke variant delegates just as you instantiate and invoke invariant delegates. In the following example, the delegate is instantiated by a lambda expression.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl21_code');" \o "Copy Code)

Dim dvariant As DVariant(Of String, String) = Function(str) str + " "

dvariant("test")

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl22_code');" \o "Copy Code)

DVariant<String, String> dvariant = (String str) => str + " ";

dvariant("test");

**Combining Variant Generic Delegates**

You should not combine variant delegates. The [Combine](http://msdn.microsoft.com/en-us/library/system.delegate.combine.aspx) method does not support variant delegate conversion and expects delegates to be of exactly the same type. This can lead to a run-time exception when you combine delegates either by using the [Combine](http://msdn.microsoft.com/en-us/library/system.delegate.combine.aspx) method (in C# and Visual Basic) or by using the **+** operator (in C#), as shown in the following code example.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl25_code');" \o "Copy Code)

Dim actObj As Action(Of Object) = Sub(x) Console.WriteLine("object: {0}", x)

Dim actStr As Action(Of String) = Sub(x) Console.WriteLine("string: {0}", x)

' The following statement throws an exception at run time.

' Dim actCombine = [Delegate].Combine(actStr, actObj)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl68_ctl00_ctl26_code');" \o "Copy Code)

Action<object> actObj = x => Console.WriteLine("object: {0}", x);

Action<string> actStr = x => Console.WriteLine("string: {0}", x);

// All of the following statements throw exceptions at run time.

// Action<string> actCombine = actStr + actObj;

// actStr += actObj;

// Delegate.Combine(actStr, actObj);

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifVariance in Generic Type Parameters for Value and Reference Types

Variance for generic type parameters is supported for reference types only. For example, DVariant<int> (DVariant(Of Int) in Visual Basic) cannot be implicitly converted to DVariant<Object> or DVaraint<long> (DVariant(Of Object) or DVaraint(Of Long) in Visual Basic), because integer is a value type.

The following example demonstrates that variance in generic type parameters is not supported for value types.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl69_ctl00_ctl00_code');" \o "Copy Code)

' The type T is covariant.

Public Delegate Function DVariant(Of Out T)() As T

' The type T is invariant.

Public Delegate Function DInvariant(Of T)() As T

Sub Test()

Dim i As Integer = 0

Dim dInt As DInvariant(Of Integer) = Function() i

Dim dVaraintInt As DVariant(Of Integer) = Function() i

' All of the following statements generate a compiler error

' because type variance in generic parameters is not supported

' for value types, even if generic type parameters are declared variant.

' Dim dObject As DInvariant(Of Object) = dInt

' Dim dLong As DInvariant(Of Long) = dInt

' Dim dVaraintObject As DInvariant(Of Object) = dInt

' Dim dVaraintLong As DInvariant(Of Long) = dInt

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl69_ctl00_ctl01_code');" \o "Copy Code)

// The type T is covariant.

public delegate T DVariant<out T>();

// The type T is invariant.

public delegate T DInvariant<T>();

public static void Test()

{

int i = 0;

DInvariant<int> dInt = () => i;

DVariant<int> dVariantInt = () => i;

// All of the following statements generate a compiler error

// because type variance in generic parameters is not supported

// for value types, even if generic type parameters are declared variant.

// DInvariant<Object> dObject = dInt;

// DInvariant<long> dLong = dInt;

// DVariant<Object> dVariantObject = dVariantInt;

// DVariant<long> dVariantLong = dVariantInt;

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRelaxed Delegate Conversion in Visual Basic

Relaxed delegate conversion, introduced in Visual Basic 2008, enables more flexibility in matching method signatures with delegate types. For example, it lets you omit parameter specifications and omit function return values when you assign a method to a delegate. For more information, see [Relaxed Delegate Conversion (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb531336.aspx).