Visual Studio 2010

**Variance in Generic Interfaces (C# and Visual Basic)**

.NET Framework 4 introduces variance support for several existing generic interfaces. Variance support enables implicit conversion of classes that implement these interfaces. The following interfaces are now variant:

* [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) (T is covariant)
* [IEnumerator<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/78dfe2yb.aspx) (T is covariant)
* [IQueryable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/bb351562.aspx) (T is covariant)
* [IGrouping<(Of <(TKey, TElement>)>)](http://msdn.microsoft.com/en-us/library/bb344977.aspx) (*TKey* and *TElement* are covariant)
* [IComparer<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/8ehhxeaf.aspx) (T is contravariant)
* [IEqualityComparer<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/ms132151.aspx) (T is contravariant)
* [IComparable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/4d7sx9hd.aspx) (T is contravariant)

Covariance permits a method to have a more derived return type than that defined by the generic type parameter of the interface. To illustrate the covariance feature, consider these generic interfaces: IEnumerable<Object> and IEnumerable<String> (IEnumerable(Of Object) and IEnumerable(Of String) in Visual Basic). The IEnumerable<String> (IEnumerable(Of String) in Visual Basic) interface does not inherit the IEnumerable<Object> interface (IEnumerable(Of Object) in Visual Basic). However, the String type does inherit the Object type, and in some cases you may want to assign objects of these interfaces to each other. This is shown in the following code example.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl22_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_ctl23_code');" \o "Copy Code)

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

IEnumerable<Object> objects = strings;

In earlier versions of the .NET Framework, this code causes a compilation error in C# and in Visual Basic with Option Strict On. But now you can use strings instead of objects, as shown in the previous example, because the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface is covariant.

Contravariance permits a method to have argument types that are less derived than that specified by the generic parameter of the interface. To illustrate contravariance, assume that you have created a BaseComparer class to compare instances of the BaseClass class. The BaseComparer class implements the IEqualityComparer<BaseClass> interface (IEqualityComparer(Of BaseClass) in Visual Basic). Because the [IEqualityComparer<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/ms132151.aspx) interface is now contravariant, you can use BaseComparer to compare instances of classes that inherit the BaseClass class. This is shown in the following code example.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl30_code');" \o "Copy Code)

' Simple hierarchy of classes.

Class BaseClass

End Class

Class DerivedClass

Inherits BaseClass

End Class

' Comparer class.

Class BaseComparer

Implements IEqualityComparer(Of BaseClass)

Public Function Equals1(ByVal x As BaseClass,

ByVal y As BaseClass) As Boolean \_

Implements IEqualityComparer(Of BaseClass).Equals

Return (x.Equals(y))

End Function

Public Function GetHashCode1(ByVal obj As BaseClass) As Integer \_

Implements IEqualityComparer(Of BaseClass).GetHashCode

Return obj.GetHashCode

End Function

End Class

Sub Test()

Dim baseComparer As IEqualityComparer(Of BaseClass) = New BaseComparer

' Implicit conversion of IEqualityComparer(Of BaseClass) to

' IEqualityComparer(Of DerivedClass).

Dim childComparer As IEqualityComparer(Of DerivedClass) = baseComparer

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl31_code');" \o "Copy Code)

// Simple hierarchy of classes.

class BaseClass { }

class DerivedClass : BaseClass { }

// Comparer class.

class BaseComparer : IEqualityComparer<BaseClass>

{

public int GetHashCode(BaseClass baseInstance)

{

return baseInstance.GetHashCode();

}

public bool Equals(BaseClass x, BaseClass y)

{

return x == y;

}

}

class Program

{

static void Test()

{

IEqualityComparer<BaseClass> baseComparer = new BaseComparer();

// Implicit conversion of IEqualityComparer<BaseClass> to

// IEqualityComparer<DerivedClass>.

IEqualityComparer<DerivedClass> childComparer = baseComparer;

}

}

For more examples, see [Using Variance in Interfaces for Generic Collections (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd465120.aspx).

Variance in generic interfaces is supported for reference types only. Value types do not support variance. For example, IEnumerable<int> (IEnumerable(Of Integer) in Visual Basic) cannot be implicitly converted to IEnumerable<object> (IEnumerable(Of Object) in Visual Basic), because integers are represented by a value type.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl33_code');" \o "Copy Code)

Dim integers As IEnumerable(Of Integer) = New List(Of Integer)

' The following statement generates a compiler error

' with Option Strict On, because Integer is a value type.

' Dim objects As IEnumerable(Of Object) = integers

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_code');" \o "Copy Code)

IEnumerable<int> integers = new List<int>();

// The following statement generates a compiler errror,

// because int is a value type.

// IEnumerable<Object> objects = integers;

It is also important to remember that classes that implement variant interfaces are still invariant. For example, although [List<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx) implements the covariant interface [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx), you cannot implicitly convert List<Object> to List<String> (List(Of Object) to List(Of String) in Visual Basic). This is illustrated in the following code example.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl41_code');" \o "Copy Code)

' The following statement generates a compiler error

' because classes are invariant.

' Dim list As List(Of Object) = New List(Of String)

' You can use the interface object instead.

Dim listObjects As IEnumerable(Of Object) = New List(Of String)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl42_code');" \o "Copy Code)

// The following line generates a compiler error

// because classes are invariant.

// List<Object> list = new List<String>();

// You can use the interface object instead.

IEnumerable<Object> listObjects = new List<String>();