Visual Studio 2010

**Using Variance for Func and Action Generic Delegates (C# and Visual Basic)**

These examples demonstrate how to use covariance and contravariance in the **Func** and **Action** generic delegates to enable reuse of methods and provide more flexibility in your code.

For more information about covariance and contravariance, see [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.gifUsing Delegates with Covariant Type Parameters

The following example illustrates the benefits of covariance support in the generic **Func** delegates. The FindByTitle method takes a parameter of the String type and returns an object of the Employee type. However, you can assign this method to the Func<String, Person> delegate (Func(Of String, Person) in Visual Basic) because Employee inherits Person.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_ctl00_ctl00_code');" \o "Copy Code)

' Simple hierarchy of classes.

Public Class Person

End Class

Public Class Employee

Inherits Person

End Class

Class Finder

Public Shared Function FindByTitle(

ByVal title As String) As Employee

' This is a stub for a method that returns

' an employee that has the specified title.

Return New Employee

End Function

Sub Test()

' Create an instance of the delegate without using variance.

Dim findEmployee As Func(Of String, Employee) =

AddressOf FindByTitle

' The delegate expects a method to return Person,

' but you can assign it a method that returns Employee.

Dim findPerson As Func(Of String, Person) =

AddressOf FindByTitle

' You can also assign a delegate

' that returns a more derived type to a delegate

' that returns a less derived type.

findPerson = findEmployee

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_ctl00_ctl01_code');" \o "Copy Code)

// Simple hierarchy of classes.

public class Person { }

public class Employee : Person { }

class Program

{

static Employee FindByTitle(String title)

{

// This is a stub for a method that returns

// an employee that has the specified title.

return new Employee();

}

static void Test()

{

// Create an instance of the delegate without using variance.

Func<String, Employee> findEmployee = FindByTitle;

// The delegate expects a method to return Person,

// but you can assign it a method that returns Employee.

Func<String, Person> findPerson = FindByTitle;

// You can also assign a delegate

// that returns a more derived type

// to a delegate that returns a less derived type.

findPerson = findEmployee;

}

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifUsing Delegates with Contravariant Type Parameters

The following example illustrates the benefits of contravariance support in the generic Action delegates. The AddToContacts method takes a parameter of the Person type. However, you can assign this method to the Action<Employee> delegate (Action(Of Employee) in Visual Basic) because Employee inherits Person.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl13_ctl00_ctl00_code');" \o "Copy Code)

Public Class Person

End Class

Public Class Employee

Inherits Person

End Class

Class AddressBook

Shared Sub AddToContacts(ByVal person As Person)

' This method adds a Person object

' to a contact list.

End Sub

Sub Test()

' Create an instance of the delegate without using variance.

Dim addPersonToContacts As Action(Of Person) =

AddressOf AddToContacts

' The Action delegate expects

' a method that has an Employee parameter,

' but you can assign it a method that has a Person parameter

' because Employee derives from Person.

Dim addEmployeeToContacts As Action(Of Employee) =

AddressOf AddToContacts

' You can also assign a delegate

' that accepts a less derived parameter

' to a delegate that accepts a more derived parameter.

addEmployeeToContacts = addPersonToContacts

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl13_ctl00_ctl01_code');" \o "Copy Code)

public class Person { }

public class Employee : Person { }

class Program

{

static void AddToContacts(Person person)

{

// This method adds a Person object

// to a contact list.

}

static void Test()

{

// Create an instance of the delegate without using variance.

Action<Person> addPersonToContacts = AddToContacts;

// The Action delegate expects

// a method that has an Employee parameter,

// but you can assign it a method that has a Person parameter

// because Employee derives from Person.

Action<Employee> addEmployeeToContacts = AddToContacts;

// You can also assign a delegate

// that accepts a less derived parameter to a delegate

// that accepts a more derived parameter.

addEmployeeToContacts = addPersonToContacts;

}

}