Visual Studio 2010

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

When you assign a method to a delegate, *covariance* and *contravariance* provide flexibility for matching a delegate type with a method signature. Covariance permits a method to have return type that is more derived than that defined in the delegate. Contravariance permits a method that has parameter types that are less derived than those in the delegate type.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample 1: Covariance

**Description**

This example demonstrates how delegates can be used with methods that have return types that are derived from the return type in the delegate signature. The data type returned by DogsHandler is of type Dogs, which derives from the Mammals type that is defined in the delegate.

**Code**

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_ctl00_ctl00_code');" \o "Copy Code)

Class Mammals

End Class

Class Dogs

Inherits Mammals

End Class

Class Test

Public Delegate Function HandlerMethod() As Mammals

Public Shared Function MammalsHandler() As Mammals

Return Nothing

End Function

Public Shared Function DogsHandler() As Dogs

Return Nothing

End Function

Sub Test()

Dim handlerMammals As HandlerMethod = AddressOf MammalsHandler

' Covariance enables this assignment.

Dim handlerDogs As HandlerMethod = AddressOf DogsHandler

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_ctl00_ctl01_code');" \o "Copy Code)

class Mammals{}

class Dogs : Mammals{}

class Program

{

// Define the delegate.

public delegate Mammals HandlerMethod();

public static Mammals MammalsHandler()

{

return null;

}

public static Dogs DogsHandler()

{

return null;

}

static void Test()

{

HandlerMethod handlerMammals = MammalsHandler;

// Covariance enables this assignment.

HandlerMethod handlerDogs = DogsHandler;

}

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample 2: Contravariance

**Description**

This example demonstrates how delegates can be used with methods that have parameters of a type that are base types of the delegate signature parameter type. With contravariance, you can use one event handler instead of separate handlers. For example, you can create an event handler that accepts an EventArgs input parameter and use it with a Button.MouseClick event that sends a MouseEventArgs type as a parameter, and also with a TextBox.KeyDown event that sends a KeyEventArgs parameter.

**Code**

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl11_ctl00_ctl00_code');" \o "Copy Code)

' Event hander that accepts a parameter of the EventArgs type.

Private Sub MultiHandler(ByVal sender As Object,

ByVal e As System.EventArgs)

Label1.Text = DateTime.Now

End Sub

Private Sub Form1\_Load(ByVal sender As System.Object,

ByVal e As System.EventArgs) Handles MyBase.Load

' You can use a method that has an EventArgs parameter,

' although the event expects the KeyEventArgs parameter.

AddHandler Button1.KeyDown, AddressOf MultiHandler

' You can use the same method

' for the event that expects the MouseEventArgs parameter.

AddHandler Button1.MouseClick, AddressOf MultiHandler

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl11_ctl00_ctl01_code');" \o "Copy Code)

// Event hander that accepts a parameter of the EventArgs type.

private void MultiHandler(object sender, System.EventArgs e)

{

label1.Text = System.DateTime.Now.ToString();

}

public Form1()

{

InitializeComponent();

// You can use a method that has an EventArgs parameter,

// although the event expects the KeyEventArgs parameter.

this.button1.KeyDown += this.MultiHandler;

// You can use the same method

// for an event that expects the MouseEventArgs parameter.

this.button1.MouseClick += this.MultiHandler;

}