.NET Framework Class Library

**Func(TResult) Delegate**

Encapsulates a method that has no parameters and returns a value of the type specified by the *TResult* parameter.

**Namespace:** [System](http://msdn.microsoft.com/en-us/library/system.aspx)  
**Assembly:** mscorlib (in mscorlib.dll)

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

Visual Basic

Public Delegate Function Func(Of Out TResult) As TResult

C#

public delegate TResult Func<out TResult>()

Visual C++

generic<typename TResult>

public delegate TResult Func()

F#

type Func =

delegate of unit -> 'TResult

**Type Parameters**

**out** **Out** **out** **out** *TResult*

The type of the return value of the method that this delegate encapsulates.

This type parameter is covariant. That is, you can use either the type you specified or any type that is more derived. For more information about covariance and contravariance, see [Covariance and Contravariance in Generics](http://msdn.microsoft.com/en-us/library/dd799517.aspx).

**Return Value**

Type: **TResult**  
The return value of the method that this delegate encapsulates.

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

You can use this delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate. The encapsulated method must correspond to the method signature that is defined by this delegate. This means that the encapsulated method must have no parameters and must return a value.

|  |
| --- |
| **Description: NoteNote** |
| To reference a method that has no parameters and returns **void** (or in Visual Basic, that is declared as a **Sub** rather than as a **Function**), use the [Action](http://msdn.microsoft.com/en-us/library/system.action.aspx) delegate instead. |

When you use the **Func<(Of <(TResult>)>)** delegate, you do not have to explicitly define a delegate that encapsulates a parameterless method. For example, the following code explicitly declares a delegate named WriteMethod and assigns a reference to the OutputTarget.SendToFile instance method to its delegate instance.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl04_code');" \o "Copy Code)

Imports System.IO

Delegate Function WriteMethod As Boolean

Module TestDelegate

Public Sub Main()

Dim output As New OutputTarget()

Dim methodCall As WriteMethod = AddressOf output.SendToFile

If methodCall() Then

Console.WriteLine("Success!")

Else

Console.WriteLine("File write operation failed.")

End If

End Sub

End Module

Public Class OutputTarget

Public Function SendToFile() As Boolean

Try

Dim fn As String = Path.GetTempFileName

Dim sw As StreamWriter = New StreamWriter(fn)

sw.WriteLine("Hello, World!")

sw.Close

Return True

Catch

Return False

End Try

End Function

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl05_code');" \o "Copy Code)

using System;

using System.IO;

delegate bool WriteMethod();

public class TestDelegate

{

public static void Main()

{

OutputTarget output = new OutputTarget();

WriteMethod methodCall = output.SendToFile;

if (methodCall())

Console.WriteLine("Success!");

else

Console.WriteLine("File write operation failed.");

}

}

public class OutputTarget

{

public bool SendToFile()

{

try

{

string fn = Path.GetTempFileName();

StreamWriter sw = new StreamWriter(fn);

sw.WriteLine("Hello, World!");

sw.Close();

return true;

}

catch

{

return false;

}

}

}

The following example simplifies this code by instantiating the **Func<(Of <(TResult>)>)** delegate instead of explicitly defining a new delegate and assigning a named method to it.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl08_code');" \o "Copy Code)

Imports System.IO

Module TestDelegate

Public Sub Main()

Dim output As New OutputTarget()

Dim methodCall As Func(Of Boolean) = AddressOf output.SendToFile

If methodCall() Then

Console.WriteLine("Success!")

Else

Console.WriteLine("File write operation failed.")

End If

End Sub

End Module

Public Class OutputTarget

Public Function SendToFile() As Boolean

Try

Dim fn As String = Path.GetTempFileName

Dim sw As StreamWriter = New StreamWriter(fn)

sw.WriteLine("Hello, World!")

sw.Close

Return True

Catch

Return False

End Try

End Function

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl09_code');" \o "Copy Code)

using System;

using System.IO;

public class TestDelegate

{

public static void Main()

{

OutputTarget output = new OutputTarget();

Func<bool> methodCall = output.SendToFile;

if (methodCall())

Console.WriteLine("Success!");

else

Console.WriteLine("File write operation failed.");

}

}

public class OutputTarget

{

public bool SendToFile()

{

try

{

string fn = Path.GetTempFileName();

StreamWriter sw = new StreamWriter(fn);

sw.WriteLine("Hello, World!");

sw.Close();

return true;

}

catch

{

return false;

}

}

}

You can use the **Func<(Of <(TResult>)>)** delegate with anonymous methods in C#, as the following example illustrates. (For an introduction to anonymous methods, see [Anonymous Methods (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/0yw3tz5k.aspx).)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl13_code');" \o "Copy Code)

using System;

using System.IO;

public class Anonymous

{

public static void Main()

{

OutputTarget output = new OutputTarget();

Func<bool> methodCall = delegate() { return output.SendToFile(); };

if (methodCall())

Console.WriteLine("Success!");

else

Console.WriteLine("File write operation failed.");

}

}

public class OutputTarget

{

public bool SendToFile()

{

try

{

string fn = Path.GetTempFileName();

StreamWriter sw = new StreamWriter(fn);

sw.WriteLine("Hello, World!");

sw.Close();

return true;

}

catch

{

return false;

}

}

}

You can also assign a lambda expression to a [Func<(Of <(T, TResult>)>)](http://msdn.microsoft.com/en-us/library/bb549151.aspx) delegate, as the following example illustrates. (For an introduction to lambda expressions, see [Lambda Expressions (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb531253.aspx) and [Lambda Expressions (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb397687.aspx).)

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl17_code');" \o "Copy Code)

Imports System.IO

Module TestDelegate

Public Sub Main()

Dim output As New OutputTarget()

Dim methodCall As Func(Of Boolean) = Function() output.SendToFile()

If methodCall() Then

Console.WriteLine("Success!")

Else

Console.WriteLine("File write operation failed.")

End If

End Sub

End Module

Public Class OutputTarget

Public Function SendToFile() As Boolean

Try

Dim fn As String = Path.GetTempFileName

Dim sw As StreamWriter = New StreamWriter(fn)

sw.WriteLine("Hello, World!")

sw.Close

Return True

Catch

Return False

End Try

End Function

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl18_code');" \o "Copy Code)

using System;

using System.IO;

public class Anonymous

{

public static void Main()

{

OutputTarget output = new OutputTarget();

Func<bool> methodCall = () => output.SendToFile();

if (methodCall())

Console.WriteLine("Success!");

else

Console.WriteLine("File write operation failed.");

}

}

public class OutputTarget

{

public bool SendToFile()

{

try

{

string fn = Path.GetTempFileName();

StreamWriter sw = new StreamWriter(fn);

sw.WriteLine("Hello, World!");

sw.Close();

return true;

}

catch

{

return false;

}

}

}

The underlying type of a lambda expression is one of the generic **Func** delegates. This makes it possible to pass a lambda expression as a parameter without explicitly assigning it to a delegate. In particular, because many methods of types in the [System.Linq](http://msdn.microsoft.com/en-us/library/system.linq.aspx) namespace have **Func** parameters, you can pass these methods a lambda expression without explicitly instantiating a **Func** delegate.

If you have an expensive computation that you want to execute only if the result is actually needed, you can assign the expensive function to a **Func<(Of <(TResult>)>)** delegate. The execution of the function can then be delayed until a property that accesses the value is used in an expression. The example in the next section demonstrates how to do this.

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

The following example demonstrates how to use a delegate that takes no parameters. This code creates a generic class named LazyValue that has a field of type **Func<(Of <(TResult>)>)**. This delegate field can store a reference to any function that returns a value of the type that corresponds to the type parameter of the LazyValue object. The LazyValue type also has a Value property that executes the function (if it has not already been executed) and returns the resulting value.

The example creates two methods and instantiates two LazyValue objects with lambda expressions that call these methods. The lambda expressions do not take parameters because they just need to call a method. As the output shows, the two methods are executed only when the value of each LazyValue object is retrieved.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl02_code');" \o "Copy Code)

Public Module Func

Public Sub Main()

' Note that each lambda expression has no parameters.

Dim lazyOne As New LazyValue(Of Integer)(Function() ExpensiveOne())

Dim lazyTwo As New LazyValue(Of Long)(Function() ExpensiveTwo("apple"))

Console.WriteLine("LazyValue objects have been created.")

' Get the values of the LazyValue objects.

Console.WriteLine(lazyOne.Value)

Console.WriteLine(lazyTwo.Value)

End Sub

Public Function ExpensiveOne() As Integer

Console.WriteLine()

Console.WriteLine("ExpensiveOne() is executing.")

Return 1

End Function

Public Function ExpensiveTwo(input As String) As Long

Console.WriteLine()

Console.WriteLine("ExpensiveTwo() is executing.")

Return input.Length

End Function

End Module

Public Class LazyValue(Of T As Structure)

Private val As Nullable(Of T)

Private getValue As Func(Of T)

' Constructor.

Public Sub New(func As Func(Of T))

Me.val = Nothing

Me.getValue = func

End Sub

Public ReadOnly Property Value() As T

Get

If Me.val Is Nothing Then

' Execute the delegate.

Me.val = Me.getValue()

End If

Return CType(val, T)

End Get

End Property

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl03_code');" \o "Copy Code)

using System;

static class Func1

{

public static void Main()

{

// Note that each lambda expression has no parameters.

LazyValue<int> lazyOne = new LazyValue<int>(() => ExpensiveOne());

LazyValue<long> lazyTwo = new LazyValue<long>(() => ExpensiveTwo("apple"));

Console.WriteLine("LazyValue objects have been created.");

// Get the values of the LazyValue objects.

Console.WriteLine(lazyOne.Value);

Console.WriteLine(lazyTwo.Value);

}

static int ExpensiveOne()

{

Console.WriteLine("\nExpensiveOne() is executing.");

return 1;

}

static long ExpensiveTwo(string input)

{

Console.WriteLine("\nExpensiveTwo() is executing.");

return (long)input.Length;

}

}

class LazyValue<T> where T : struct

{

private Nullable<T> val;

private Func<T> getValue;

// Constructor.

public LazyValue(Func<T> func)

{

val = null;

getValue = func;

}

public T Value

{

get

{

if (val == null)

// Execute the delegate.

val = getValue();

return (T)val;

}

}

}

/\* The example produces the following output:

LazyValue objects have been created.

ExpensiveOne() is executing.

1

ExpensiveTwo() is executing.

5

\*/