Visual Studio 2010

**Expression Trees (C# and Visual Basic)**

Expression trees represent code in a tree-like data structure, where each node is an expression, for example, a method call or a binary operation such as x < y.

You can compile and run code represented by expression trees. This enables dynamic modification of executable code, the execution of LINQ queries in various databases, and the creation of dynamic queries. For more information about expression trees in LINQ, see [How to: Use Expression Trees to Build Dynamic Queries (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/bb882637.aspx) and [Walkthrough: Creating an IQueryable LINQ Provider](http://msdn.microsoft.com/en-us/library/bb546158.aspx).

Expression trees are also used in the dynamic language runtime (DLR) to provide interoperability between dynamic languages and the .NET Framework and to enable compiler writers to emit expression trees instead of Microsoft intermediate language (MSIL). For more information about the DLR, see [Dynamic Language Runtime Overview](http://msdn.microsoft.com/en-us/library/dd233052.aspx).

You can have the C# or Visual Basic compiler create an expression tree for you based on an anonymous lambda expression, or you can create expression trees manually by using the [System.Linq.Expressions](http://msdn.microsoft.com/en-us/library/system.linq.expressions.aspx) namespace.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating Expression Trees from Lambda Expressions

When a lambda expression is assigned to a variable of type [Expression<(Of <(TDelegate>)>)](http://msdn.microsoft.com/en-us/library/bb335710.aspx), the compiler emits code to build an expression tree that represents the lambda expression.

The C# and Visual Basic compilers can generate expression trees only from expression lambdas (or single-line lambdas). It cannot parse statement lambdas (or multi-line lambdas). For more information about lambda expressions in C#, see [Lambda Expressions (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb397687.aspx); for Visual Basic, see [Lambda Expressions (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb531253.aspx).

The following code examples demonstrate how to have the C# and Visual Basic compilers create an expression tree that represents the lambda expression num => num < 5 (C#) or Function(num) num < 5 (Visual Basic).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl03_code');" \o "Copy Code)

Dim lambda As Expression(Of Func(Of Integer, Boolean)) =

Function(num) num < 5

C#

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

Expression<Func<int, bool>> lambda = num => num < 5;

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating Expression Trees by Using the API

To create expression trees by using the API, use the [Expression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.expression.aspx) class. This class contains static factory methods that create expression tree nodes of specific types, for example, [ParameterExpression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.parameterexpression.aspx), which represents a variable or parameter, or [MethodCallExpression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.methodcallexpression.aspx), which represents a method call. [ParameterExpression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.parameterexpression.aspx), [MethodCallExpression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.methodcallexpression.aspx), and the other expression-specific types are also defined in the [System.Linq.Expressions](http://msdn.microsoft.com/en-us/library/system.linq.expressions.aspx) namespace. These types derive from the abstract type [Expression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.expression.aspx).

The following code example demonstrates how to create an expression tree that represents the lambda expression num => num < 5 (C#) or Function(num) num < 5 (Visual Basic) by using the API.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl07_code');" \o "Copy Code)

' Import the following namespace to your project: System.Linq.Expressions

' Manually build the expression tree for the lambda expression num => num < 5.

Dim numParam As ParameterExpression = Expression.Parameter(GetType(Integer), "num")

Dim five As ConstantExpression = Expression.Constant(5, GetType(Integer))

Dim numLessThanFive As BinaryExpression = Expression.LessThan(numParam, five)

Dim lambda1 As Expression(Of Func(Of Integer, Boolean)) =

Expression.Lambda(Of Func(Of Integer, Boolean))(

numLessThanFive,

New ParameterExpression() {numParam})

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl08_code');" \o "Copy Code)

// Add the following using directive to your code file:

// using System.Linq.Expressions;

// Manually build the expression tree for

// the lambda expression num => num < 5.

ParameterExpression numParam = Expression.Parameter(typeof(int), "num");

ConstantExpression five = Expression.Constant(5, typeof(int));

BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);

Expression<Func<int, bool>> lambda1 =

Expression.Lambda<Func<int, bool>>(

numLessThanFive,

new ParameterExpression[] { numParam });

In .NET Framework 4, the expression trees API also supports assignments and control flow expressions such as loops, conditional blocks, and **try-catch** blocks. By using the API, you can create expression trees that are more complex than those that can be created from lambda expressions by the C# and Visual Basic compilers. The following example demonstrates how to create an expression tree that calculates the factorial of a number.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl09_code');" \o "Copy Code)

' Creating a parameter expression.

Dim value As ParameterExpression =

Expression.Parameter(GetType(Integer), "value")

' Creating an expression to hold a local variable.

Dim result As ParameterExpression =

Expression.Parameter(GetType(Integer), "result")

' Creating a label to jump to from a loop.

Dim label As LabelTarget = Expression.Label(GetType(Integer))

' Creating a method body.

Dim block As BlockExpression = Expression.Block(

New ParameterExpression() {result},

Expression.Assign(result, Expression.Constant(1)),

Expression.Loop(

Expression.IfThenElse(

Expression.GreaterThan(value, Expression.Constant(1)),

Expression.MultiplyAssign(result,

Expression.PostDecrementAssign(value)),

Expression.Break(label, result)

),

label

)

)

' Compile an expression tree and return a delegate.

Dim factorial As Integer =

Expression.Lambda(Of Func(Of Integer, Integer))(block, value).Compile()(5)

Console.WriteLine(factorial)

' Prints 120.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl10_code');" \o "Copy Code)

// Creating a parameter expression.

ParameterExpression value = Expression.Parameter(typeof(int), "value");

// Creating an expression to hold a local variable.

ParameterExpression result = Expression.Parameter(typeof(int), "result");

// Creating a label to jump to from a loop.

LabelTarget label = Expression.Label(typeof(int));

// Creating a method body.

BlockExpression block = Expression.Block(

// Adding a local variable.

new[] { result },

// Assigning a constant to a local variable: result = 1

Expression.Assign(result, Expression.Constant(1)),

// Adding a loop.

Expression.Loop(

// Adding a conditional block into the loop.

Expression.IfThenElse(

// Condition: value > 1

Expression.GreaterThan(value, Expression.Constant(1)),

// If true: result \*= value --

Expression.MultiplyAssign(result,

Expression.PostDecrementAssign(value)),

// If false, exit the loop and go to the label.

Expression.Break(label, result)

),

// Label to jump to.

label

)

);

// Compile and execute an expression tree.

int factorial = Expression.Lambda<Func<int, int>>(block, value).Compile()(5);

Console.WriteLine(factorial);

// Prints 120.

For more information, see [Generating Dynamic Methods with Expression Trees in Visual Studio 2010](http://go.microsoft.com/fwlink/?LinkId=169513).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifParsing Expression Trees

The following code example demonstrates how the expression tree that represents the lambda expression num => num < 5 (C#) or Function(num) num < 5 (Visual Basic) can be decomposed into its parts.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl55_ctl00_ctl00_code');" \o "Copy Code)

' Import the following namespace to your project: System.Linq.Expressions

' Create an expression tree.

Dim exprTree As Expression(Of Func(Of Integer, Boolean)) = Function(num) num < 5

' Decompose the expression tree.

Dim param As ParameterExpression = exprTree.Parameters(0)

Dim operation As BinaryExpression = exprTree.Body

Dim left As ParameterExpression = operation.Left

Dim right As ConstantExpression = operation.Right

Console.WriteLine(String.Format("Decomposed expression: {0} => {1} {2} {3}",

param.Name, left.Name, operation.NodeType, right.Value))

' This code produces the following output:

'

' Decomposed expression: num => num LessThan 5

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl55_ctl00_ctl01_code');" \o "Copy Code)

// Add the following using directive to your code file:

// using System.Linq.Expressions;

// Create an expression tree.

Expression<Func<int, bool>> exprTree = num => num < 5;

// Decompose the expression tree.

ParameterExpression param = (ParameterExpression)exprTree.Parameters[0];

BinaryExpression operation = (BinaryExpression)exprTree.Body;

ParameterExpression left = (ParameterExpression)operation.Left;

ConstantExpression right = (ConstantExpression)operation.Right;

Console.WriteLine("Decomposed expression: {0} => {1} {2} {3}",

param.Name, left.Name, operation.NodeType, right.Value);

// This code produces the following output:

// Decomposed expression: num => num LessThan 5

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifImmutability of Expression Trees

Expression trees should be immutable. This means that if you want to modify an expression tree, you must construct a new expression tree by copying the existing one and replacing nodes in it. You can use an expression tree visitor to traverse the existing expression tree. For more information, see [How to: Modify Expression Trees (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/bb546136.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCompiling Expression Trees

The [Expression<(Of <(TDelegate>)>)](http://msdn.microsoft.com/en-us/library/bb335710.aspx) type provides the [Compile](http://msdn.microsoft.com/en-us/library/dd294259.aspx) method that compiles the code represented by an expression tree into an executable delegate.

The following code example demonstrates how to compile an expression tree and run the resulting code.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl57_ctl00_ctl02_code');" \o "Copy Code)

' Creating an expression tree.

Dim expr As Expression(Of Func(Of Integer, Boolean)) =

Function(num) num < 5

' Compiling the expression tree into a delegate.

Dim result As Func(Of Integer, Boolean) = expr.Compile()

' Invoking the delegate and writing the result to the console.

Console.WriteLine(result(4))

' Prints True.

' You can also use simplified syntax

' to compile and run an expression tree.

' The following line can replace two previous statements.

Console.WriteLine(expr.Compile()(4))

' Also prints True.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl57_ctl00_ctl03_code');" \o "Copy Code)

// Creating an expression tree.

Expression<Func<int, bool>> expr = num => num < 5;

// Compiling the expression tree into a delegate.

Func<int, bool> result = expr.Compile();

// Invoking the delegate and writing the result to the console.

Console.WriteLine(result(4));

// Prints True.

// You can also use simplified syntax

// to compile and run an expression tree.

// The following line can replace two previous statements.

Console.WriteLine(expr.Compile()(4));

// Also prints True.

Visual Studio 2010

**How to: Execute Expression Trees (C# and Visual Basic)**

This topic shows you how to execute an expression tree. Executing an expression tree may return a value, or it may just perform an action such as calling a method.

Only expression trees that represent lambda expressions can be executed. Expression trees that represent lambda expressions are of type [LambdaExpression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.lambdaexpression.aspx) or [Expression<(Of <(TDelegate>)>)](http://msdn.microsoft.com/en-us/library/bb335710.aspx). To execute these expression trees, call the [Compile](http://msdn.microsoft.com/en-us/library/system.linq.expressions.lambdaexpression.compile.aspx) method to create an executable delegate, and then invoke the delegate.

|  |
| --- |
| **NoteNote** |
| If the type of the delegate is not known, that is, the lambda expression is of type [LambdaExpression](http://msdn.microsoft.com/en-us/library/system.linq.expressions.lambdaexpression.aspx) and not [Expression<(Of <(TDelegate>)>)](http://msdn.microsoft.com/en-us/library/bb335710.aspx), you must call the [DynamicInvoke](http://msdn.microsoft.com/en-us/library/system.delegate.dynamicinvoke.aspx) method on the delegate instead of invoking it directly. |

If an expression tree does not represent a lambda expression, you can create a new lambda expression that has the original expression tree as its body, by calling the [Lambda<(Of <(TDelegate>)>)(Expression, IEnumerable<(Of <(ParameterExpression>)>))](http://msdn.microsoft.com/en-us/library/bb340145.aspx) method. Then, you can execute the lambda expression as described earlier in this section.

Example

The following code example demonstrates how to execute an expression tree that represents raising a number to a power by creating a lambda expression and executing it. The result, which represents the number raised to the power, is displayed.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl31_ctl00_ctl00_code');" \o "Copy Code)

' The expression tree to execute.

Dim be As BinaryExpression = Expression.Power(Expression.Constant(2.0R), Expression.Constant(3.0R))

' Create a lambda expression.

Dim le As Expression(Of Func(Of Double)) = Expression.Lambda(Of Func(Of Double))(be)

' Compile the lambda expression.

Dim compiledExpression As Func(Of Double) = le.Compile()

' Execute the lambda expression.

Dim result As Double = compiledExpression()

' Display the result.

MsgBox(result)

' This code produces the following output:

' 8

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl31_ctl00_ctl01_code');" \o "Copy Code)

// The expression tree to execute.

BinaryExpression be = Expression.Power(Expression.Constant(2D), Expression.Constant(3D));

// Create a lambda expression.

Expression<Func<double>> le = Expression.Lambda<Func<double>>(be);

// Compile the lambda expression.

Func<double> compiledExpression = le.Compile();

// Execute the lambda expression.

double result = compiledExpression();

// Display the result.

Console.WriteLine(result);

// This code produces the following output:

// 8

Compiling the Code

* Add a project reference to System.Core.dll if it is not already referenced.
* Include the System.Linq.Expressions namespace.