Expression Cheat Sheet

ExpressionType	Type	Example
Add	BinaryExpression	`int i = 2, j = 3;` `Expression<Func<int>> example = () => i + j;`
AddChecked	BinaryExpression	`int i = Int32.MaxValue, j = 1;` `Expression<Func<int>> example = () => checked(i + j);`

And	BinaryExpression	`Dim i As Boolean = True, j As Boolean = False`
		`Dim sample As Expression(Of Func(Of Boolean)) = _`
		`  Function() i And j`

AndAlso	BinaryExpression	`bool i = true, j = false;` `Expression<Func<bool>> example = () => i && j;`
ArrayLength	UnaryExpression	`int[] values = {1, 2, 3};` `Expression<Func<int>> example = () => values.Length;`
ArrayIndex	MethodCallExpression	`int[] values = {1, 2, 3};` `Expression<Func<int>> example = () => values[1];`
Call	MethodCallExpression	`var sample = new Sample();` `Expression<Func<int>> example = () => sample.Calc();`
Coalesce	BinaryExpression	`int? i = null, j = 5;` `Expression<Func<int?>> example = () => i ?? j;`

Conditional	ConditionalExpression	`int i = 3, j = 5;`
		`bool k = false;`
		`Expression<Func<int?>> example = () => k ? i : j;`

Constant	ConstantExpression	`Expression<Func<int>> example = () => 5;`

Convert	UnaryExpression	`int i = 5;`
		`object j = i;`
		`Expression<Func<int>> example = () => (int) j;`

ConvertChecked	UnaryExpression	`long i = 5;` `Expression<Func<int>> example = () => checked((int)i);`
Divide	BinaryExpression	`int i = 21, j = 3;` `Expression<Func<int>> example = () => i / j;`
Equal	BinaryExpression	`int i = 21, j = 3;` `Expression<Func<bool>> example = () => i == j;`
ExclusiveOr	BinaryExpression	`int i = 12, j = 7;` `Expression<Func<int>> example = () => i ^ j;`
GreaterThan	BinaryExpression	`int i = 12, j = 7;`

		`Expression<Func<bool>> example = () => i > j;`
GreaterThanOrEqual	BinaryExpression	`int i = 12, j = 7;` `Expression<Func<bool>> example = () => i >= j;`

Invoke	InvocationExpression	`Expression<Func<int, int, int>> expr =`
		`(i, j) => i + j;`
		`Expression invoke = Expression.Invoke(`
		`expr,`
		`Expression.Constant(5),`
		`Expression.Constant(4));`
		`Expression<Func<int>> example =`
		`Expression.Lambda<Func<int>>(invoke);`

Lambda	LambdaExpression	`Expression<Func<int>> example =` `Expression.Lambda<Func<int>>(Expression.Constant(5));`
LeftShift	BinaryExpression	`int i = 8;` `Expression<Func<int>> example = () => i << 1;`
LessThan	BinaryExpression	`int i = 12, j = 7;` `Expression<Func<bool>> example = () => i < j;`
LessThanOrEqual	BinaryExpression	`int i = 12, j = 7;` `Expression<Func<bool>> example = () => i <= j;`
ListInit	ListInitExpression	`Expression<Func<List<int>>> example =` `() => new List<int> {1, 2, 3};`
MemberAccess	MemberExpression	`var c = new Customer {Name = "Bob"};` `Expression<Func<string>> example = () => c.Name;`
MemberInit	MemberInitExpression	`Expression<Func<Customer>> example =` `() => new Customer {Name = "Bob"};`
Modulo	BinaryExpression	`int i = 5, j = 3;` `Expression<Func<int>> example = () => i % j;`
Multiply	BinaryExpression	`int i = 5, j = 3;` `Expression<Func<int>> example = () => i * j;`
MultiplyChecked	BinaryExpression	`int i = 5, j = 3;` `Expression<Func<int>> example = () => checked(i * j);`
Negate	UnaryExpression	`int i = 5;` `Expression<Func<int>> example = () => -i;`
UnaryPlus	UnaryExpression	`var m = new Money { Amount = -10m };` `Expression<Func<Money>> example = () => +m;`
NegateChecked	UnaryExpression	`int i = 5;` `Expression<Func<int>> example = () => checked(-i);`
New	NewExpression	`Expression<Func<Customer>> example =` `() => new Customer();`

NewArrayInit	NewArrayExpression	`Expression<Func<int[]>> example =` `  () => new[] {1, 2, 3};`
NewArrayBounds	NewArrayExpression	`Expression<Func<int[]>> example = () => new int[10];`
Not	UnaryExpression	`bool val = true;` `Expression<Func<bool>> example = () => !val;`
NotEqual	BinaryExpression	`int i = 4, j = 7;` `Expression<Func<bool>> example = () => i != j;`

Or	BinaryExpression	`Dim i As Boolean = True, j As Boolean = False`
		`Dim sample As Expression(Of Func(Of Boolean)) = _`
		` Function() i Or j`

OrElse	BinaryExpression	`bool i = true, j = false;` `Expression<Func<bool>> example = () => i || j;`

Parameter	ParameterExpression	`// (i, j) => i + j;`
		`ParameterExpression param1 =`
		`  Expression.Parameter(typeof (int), "i");`
		`ParameterExpression param2 =`
		`  Expression.Parameter(typeof (int), "j");`
		`var addExpression = Expression.Add(param1, param2);`
		`var example = Expression.Lambda<Func<int, int, int>>(`
		`  addExpression, param1, param2);`

Power	BinaryExpression	`Dim i As Integer = 3, j As Integer = 2`
		`Dim sample As Expression(Of Func(Of Integer)) = _`
		` Function() i ^ j`

Quote	UnaryExpression	`int i = 3, j = 2;`
		`Expression<Func<int>> inner = () => i * j;`
		`var quoted = Expression.Quote(inner);`
		`Expression<Func<Expression<Func<int>>>> example =`
		` Expression.Lambda<Func<Expression<Func<int>>>>(quoted);`

RightShift	BinaryExpression	`int i = 8;` `Expression<Func<int>> example = () => i >> 1;`
Subtract	BinaryExpression	`int i = 8, j = 5;` `Expression<Func<int>> example = () => i - j;`
SubtractChecked	BinaryExpression	`int i = 8, j = 5;` `Expression<Func<int>> example = () => checked(i - j);`
TypeAs	UnaryExpression	`var c = new Customer {Name = "Bob"};` `Expression<Func<Person>> example = () => c as Person;`
TypeIs	TypeBinaryExpression	`var c = new Customer {Name = "Bob"};` `Expression<Func<bool>> example = () => c is int;`