



# Visual Lexicon of LINQ

LINQ extends the C# language with native data querying capabilities giving you SQL-like expressiveness in C# (and other .NET languages). LINQ can be applied to in-memory data (variables), XML, databases, and more, limited only by the LINQ providers you have on hand. This wallchart is a companion to the Simple Talk article **A Visual Lexicon of LINQ**, which provides a visual example for each LINQ operator to provide a quick understanding of how each one conveys its input to its output. An example is shown immediately below. Thanks to OzCode (<https://oz-code.com/>) for the visual pattern renderings.



This example visualization of the **Count** operator shows how some of the 7 input elements are included and some excluded, and how the included elements collapse to a single output element (the “Collapse all to one” pattern). Purple lines (/) simply indicate an item is *selected* in the editor; other patterns also show grey lines (/) for *unselected* elements.

## Characteristics of each LINQ operator

Category	Operator	Position		Syntax		Execution		Laziness		Time: O(x)	Space: O(x)	Complexity	Optional Features	
		Initial	Final	Lambda	Query	Deferred	Immediate	Some elements	All elements				Available index	Input Transform
Aggregate	Aggregate											n 1		
	Average											n 1		
	Count											1   n 1		
	LongCount											1   n 1		
	Max											n 1		
	Min											n 1		
	Sum											n 1		
Conversion	AsEnumerable											n 1		
	Cast											n 1		
	OfType											n 1		
	ToArray											n n		
	ToDictionary											n n		
	ToList											n n		
	ToLookup											n n		
Elements	ElementAt											1   n 1		
	ElementAtOrDefault											1   n 1		
	First											1   n 1		
	FirstOrDefault											1   n 1		
	Last											1   n 1		
	LastOrDefault											1   n 1		
	Single											1   n 1		
	SingleOrDefault											1   n 1		
Generation	DefaultIfEmpty											n 1		
	Empty											1 1		
	Range											n 1		
	Repeat											n 1		
Group	GroupBy											n n		
Join	Concat											n n		
	GroupJoin											n n		
	Join											n n		
	Zip											n 1		
Ordering	OrderBy											n*logn n		
	OrderByDescending											n*logn n		
	Reverse											n n		
	ThenBy											n*logn n		
	ThenByDescending											n*logn n		
Partitioning	Skip											n 1		
	SkipWhile											n 1		
	Take											n 1		
	TakeWhile											n 1		
Projection	Select											n 1		
	SelectMany											n*m 1		
Quantifiers	All											n 1		
	Any											n 1		
	Contains											n 1		
	SequenceEqual											n 1		
Restriction	Where											n 1		
Sets	Distinct											n n		
	Except											n n		
	Intersect											n n		
	Union											n n		

### Position

Specifies where this operator may occur in a sequence: those that generate a sequence (a *source*) must be in **initial** position; those that transform or process a sequence are **intermediate**; those that convert the sequence to an object or a value (a *sink*) are **final**. For example, **Select** (intermediate) might appear as `op1(...).op2(...).Select(...).op3(...)` while **Count** (final) must appear as `op1(...).op2(...).Count(...)`.



### Syntax

Every operator exists in **lambda** syntax; only a select few exist in **query** syntax but those are the most commonly used; both styles may be used together. The snippet shows the same result with both styles.

```
var list = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
Func<int, bool> isOdd = (n => n % 2 == 1);
var queryResult = from n in list where isOdd(n) select n;
var lambdaResult = list.Where(isOdd);
```

### Execution and Laziness

LINQ **defers** execution for many operators; data results are returned **immediately** only for some. Further, when executing a query, only as much of a sequence that is actually needed is evaluated: that might be just the **first** element, **all** elements, or **some** number in between. This could vary for any given operator depending on arguments supplied. Ex: with no arguments **First** evaluates only the first element, but with a condition **First** might evaluate any number (or all) arguments; thus, **First** shows all 3 possibilities marked.

### Complexity

**Time complexity** specifies how long an operator takes to run. Notes:

- >> **Count** & **LongCount** run in **O(1)** if the underlying type implements **ICollection**; otherwise **O(n)**.
- >> **ElementAt(OrDefault)** & **Last(OrDefault)** run in **O(1)** if the type implements **IList<T>**; otherwise **O(n)**.
- >> **First(OrDefault)** & **Single(OrDefault)** run in **O(n)** if a condition is present; otherwise **O(1)**.

**Space complexity** specifies how much memory is used with respect to the input size.

### Optional Features

**Available index:** When processing a given element, operator may use the element's index in a computation.

**Input Transform:** Accepts a transform function for input (rather than invoking **Select** then the operator).

**Output Projection:** Accepts a projection function for output (rather than invoking the operator then **Select**).

**Custom Comparer:** Operators that do comparisons can accept a custom comparer rather than the default.

**Conditional Selection:** Accepts a filtering function for output (rather than invoking **Where** then the operator).

```
pets.Select((pet, i) => $" {i} {pet.Name}")
```

```
numbers.Sum(n => n > 5 ? n : 0)
```

```
pets.GroupBy(p => p.Age, p => p.Name)
```

```
fruits.Contains(pear, produceComparer)
```

```
words.Single(w => w.Length > minLength)
```

## Visual Patterns of LINQ Operators

LINQ operators can be categorized into these ten patterns. Note that some operators fit more than one pattern. For example, **All** fits **Collapse all to one** when returning true, but **One to one** when returning false. **Count** with conditional selection fits **Collapse some to one** but without it, fits **Collapse all to one**. Refer to the main article for details of each operator.

Category	LINQ Operators	Visual Pattern
<b>Collapse all to one</b>	Aggregate All Any Average Count LongCount Sum	
<b>Collapse some to one</b>	Count LongCount SequenceEqual	
<b>Collapse groups</b>	GroupBy ToLookup	
<b>Expand groups</b>	SelectMany	
<b>One to one</b>	All Any Contains ElementAt(OrDefault) First(OrDefault) Last(OrDefault) Max Min Single(OrDefault)	
<b>None to one</b>	DefaultIfEmpty ElementAtOrDefault FirstOrDefault LastOrDefault SingleOrDefault	
<b>None to some</b>	Range Repeat	
<b>Convey all/ order retained</b>	AsEnumerable Cast Concat DefaultIfEmpty GroupJoin Select Single Skip(While) Take(While) ToArray ToDictionary ToList Union Zip	
<b>Convey all/ order changed</b>	OrderBy(Descending) Reverse ThenBy(Descending)	
<b>Convey some</b>	Distinct Except Intersect Join OfType Skip(While) Take(While) Where Zip	

### Further Reading

- [LINQ on MSDN](#)
- [Enumerable Methods](#)
- [LINQ Debugging and Visualization](#)
- [Query Expression Syntax for Standard Query Operators](#)
- [101 LINQ Samples](#) or [LINQ Samples.com](#)