Tell us about your PDF experience.

System.Linq Namespace

Reference

Provides classes and interfaces that support queries that use Language-Integrated Query (LINQ).

Classes

Expand table

Enumerable	Provides a set of `static` (`Shared` in Visual Basic) methods for querying objects that implement IEnumerable<T>.
EnumerableExecutor	Represents an expression tree and provides functionality to execute the expression tree after rewriting it.
EnumerableExecutor<T>	Represents an expression tree and provides functionality to execute the expression tree after rewriting it.
EnumerableQuery	Represents an IEnumerable as an EnumerableQuery data source.
EnumerableQuery<T>	Represents an IEnumerable<T> collection as an IQueryable<T> data source.
ImmutableArrayExtensions	LINQ extension method overrides that offer greater efficiency for ImmutableArray<T> than the standard LINQ methods **NuGet package**: System.Collections.Immutable (about immutable collections and how to install)
Lookup<TKey,TElement>	Represents a collection of keys each mapped to one or more values.
OrderedParallelQuery<TSource>	Represents a sorted, parallel sequence.
ParallelEnumerable	Provides a set of methods for querying objects that implement ParallelQuery{TSource}. This is the parallel equivalent of Enumerable.
ParallelQuery	Represents a parallel sequence.
ParallelQuery<TSource>	Represents a parallel sequence.
Queryable	Provides a set of `static` (`Shared` in Visual Basic) methods for querying data structures that implement IQueryable<T>.

Interfaces

Expand table

IGrouping<TKey,TElement>	Represents a collection of objects that have a common key.
ILookup<TKey,TElement>	Defines an indexer, size property, and Boolean search method for data structures that map keys to IEnumerable<T> sequences of values.
IOrdered Enumerable<TElement>	Represents a sorted sequence.
IOrderedQueryable	Represents the result of a sorting operation.
IOrderedQueryable<T>	Represents the result of a sorting operation.
IQueryable	Provides functionality to evaluate queries against a specific data source wherein the type of the data is not specified.
IQueryable<T>	Provides functionality to evaluate queries against a specific data source wherein the type of the data is known.
IQueryProvider	Defines methods to create and execute queries that are described by an IQueryable object.

Enums

Expand table

Parallel Execution Mode	The query execution mode is a hint that specifies how the system should handle performance trade-offs when parallelizing queries.
Parallel Merge Options	Specifies the preferred type of *output merge* to use in a query. In other words, it indicates how PLINQ should merge the results from the various partitions back into a single result sequence. This is a hint only, and may not be respected by the system when parallelizing all queries.

Remarks

The System.Linq namespace is in the System.Core assembly (in System.Core.dll).

The Enumerable class contains LINQ standard query operators that operate on objects that implement IEnumerable<T>.

The Queryable class contains LINQ standard query operators that operate on objects that implement IQueryable<T>.

For more information, see LINQ to SQL.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enumerable Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Enumerable.cs

Provides a set of `static` (`Shared` in Visual Basic) methods for querying objects that implement IEnumerable<T>.

```
C#
public static class Enumerable
```

Inheritance Object → Enumerable

Remarks

The methods in this class provide an implementation of the standard query operators for querying data sources that implement IEnumerable<T>. The standard query operators are general purpose methods that follow the LINQ pattern and enable you to express traversal, filter, and projection operations over data in any .NET-based programming language.

The majority of the methods in this class are defined as extension methods that extend IEnumerable<T>. This means they can be called like an instance method on any object that implements IEnumerable<T>.

Methods that are used in a query that returns a sequence of values do not consume the target data until the query object is enumerated. This is known as deferred execution. Methods that are used in a query that returns a singleton value execute and consume the target data immediately.

Methods

Expand table

Aggregate<TSource,TAccumulate,TResult> (IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average(IEnumerable<Decimal>)	Computes the average of a sequence of Decimal values.
Average(IEnumerable<Double>)	Computes the average of a sequence of Double values.
Average(IEnumerable<Int32>)	Computes the average of a sequence of Int32 values.
Average(IEnumerable<Int64>)	Computes the average of a sequence of Int64 values.
Average(IEnumerable<Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values.
Average(IEnumerable<Nullable<Double>>)	Computes the average of a sequence of nullable Double

	values.
Average(IEnumerable<Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values.
Average(IEnumerable<Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values.
Average(IEnumerable<Nullable<Single>>)	Computes the average of a sequence of nullable Single values.
Average(IEnumerable<Single>)	Computes the average of a sequence of Single values.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.

Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.

ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Empty<TResult>()	Returns an empty IEnumerable<T> that has the specified type argument.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.

FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.

LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max(IEnumerable<Decimal>)	Returns the maximum value in a sequence of Decimal values.
Max(IEnumerable<Double>)	Returns the maximum value in a sequence of Double values.
Max(IEnumerable<Int32>)	Returns the maximum value in a sequence of Int32 values.
Max(IEnumerable<Int64>)	Returns the maximum value in a sequence of Int64 values.
Max(IEnumerable<Nullable<Decimal>>)	Returns the maximum value in a sequence of nullable Decimal values.
Max(IEnumerable<Nullable<Double>>)	Returns the maximum value in a sequence of nullable Double values.
Max(IEnumerable<Nullable<Int32>>)	Returns the maximum value in a sequence of nullable Int32 values.
Max(IEnumerable<Nullable<Int64>>)	Returns the maximum value in a sequence of nullable Int64 values.
Max(IEnumerable<Nullable<Single>>)	Returns the maximum value in a sequence of nullable Single values.
Max(IEnumerable<Single>)	Returns the maximum value in a sequence of Single values.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.

Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.

Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min(IEnumerable<Decimal>)	Returns the minimum value in a sequence of Decimal values.
Min(IEnumerable<Double>)	Returns the minimum value in a sequence of Double values.
Min(IEnumerable<Int32>)	Returns the minimum value in a sequence of Int32 values.
Min(IEnumerable<Int64>)	Returns the minimum value in a sequence of Int64 values.
Min(IEnumerable<Nullable<Decimal>>)	Returns the minimum value in a sequence of nullable Decimal values.
Min(IEnumerable<Nullable<Double>>)	Returns the minimum value in a sequence of nullable Double values.
Min(IEnumerable<Nullable<Int32>>)	Returns the minimum value in a sequence of nullable Int32 values.
Min(IEnumerable<Nullable<Int64>>)	Returns the minimum value in a sequence of nullable Int64 values.
Min(IEnumerable<Nullable<Single>>)	Returns the minimum value in a sequence of nullable Single values.
Min(IEnumerable<Single>)	Returns the minimum value in

	a sequence of Single values.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.

OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Range(Int32, Int32)	Generates a sequence of integral numbers within a specified range.
Repeat<TResult>(TResult, Int32)	Generates a sequence that contains one repeated value.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an

	IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more

	than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum(IEnumerable<Decimal>)	Computes the sum of a sequence of Decimal values.
Sum(IEnumerable<Double>)	Computes the sum of a sequence of Double values.

Sum(IEnumerable<Int32>)	Computes the sum of a sequence of Int32 values.
Sum(IEnumerable<Int64>)	Computes the sum of a sequence of Int64 values.
Sum(IEnumerable<Nullable<Decimal>>)	Computes the sum of a sequence of nullable Decimal values.
Sum(IEnumerable<Nullable<Double>>)	Computes the sum of a sequence of nullable Double values.
Sum(IEnumerable<Nullable<Int32>>)	Computes the sum of a sequence of nullable Int32 values.
Sum(IEnumerable<Nullable<Int64>>)	Computes the sum of a sequence of nullable Int64 values.
Sum(IEnumerable<Nullable<Single>>)	Computes the sum of a sequence of nullable Single values.
Sum(IEnumerable<Single>)	Computes the sum of a sequence of Single values.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.

Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ThenBy<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.
ThenByDescending<TSource,TKey>(IOrdered Enumerable<TSource>, Func<TSource,TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(IOrdered Enumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TKey,TValue>(IEnumerable<KeyValue Pair<TKey,TValue>>)	Creates a dictionary from an enumeration according to the

	default comparer for the key type.
ToDictionary<TKey,TValue>(IEnumerable<KeyValuePair<TKey,TValue>>, IEqualityComparer<TKey>)	Creates a dictionary from an enumeration according to specified key comparer.
ToDictionary<TKey,TValue>(IEnumerable<ValueTuple<TKey,TValue>>)	Creates a dictionary from an enumeration according to the default comparer for the key type.
ToDictionary<TKey,TValue>(IEnumerable<ValueTuple<TKey,TValue>>, IEqualityComparer<TKey>)	Creates a dictionary from an enumeration according to specified key equality comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.

ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.

UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Standard Query Operators overview (C#)
- Standard Query Operators overview (Visual Basic)
- Extension Methods (C# Programming Guide)
- Extension Methods (Visual Basic)

Enumerable.Aggregate Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.

Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)

Source: Aggregate.cs

Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.

C#

```
public static TResult Aggregate<TSource,TAccumulate,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source, TAccumulate seed,
Func<TAccumulate,TSource,TAccumulate> func, Func<TAccumulate,TResult>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

`TResult`

The type of the resulting value.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`func` Func<TAccumulate,TSource,TAccumulate>

An accumulator function to be invoked on each element.

`resultSelector` Func<TAccumulate,TResult>

A function to transform the final accumulator value into the result value.

Returns

TResult

The transformed final accumulator value.

Exceptions

ArgumentNullException

`source` or `func` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use Aggregate to apply an accumulator function and a result selector.

C#

```
string[] fruits = { "apple", "mango", "orange", "passionfruit", "grape"
};

// Determine whether any string in the array is longer than "banana".
string longestName =
    fruits.Aggregate("banana",
                     (longest, next) =>
                         next.Length > longest.Length ? next : longest,
                     // Return the final result as an upper case string.
                     fruit => fruit.ToUpper());

Console.WriteLine(
    "The fruit with the longest name is {0}.",
    longestName);

// This code produces the following output:
//
// The fruit with the longest name is PASSIONFRUIT.
```

Remarks

The Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>) method makes it simple to perform a calculation over a sequence of values. This method works by calling `func` one time for each element in `source`. Each time `func` is called, Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>) passes both the element from the sequence and an aggregated value (as the first argument to `func`). The value of the `seed` parameter is used as the initial aggregate value. The result of `func` replaces the previous aggregated value. The final result of `func` is passed to `resultSelector` to obtain the final result of Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>).

To simplify common aggregation operations, the standard query operators also include a general purpose count method, Count, and four numeric aggregation methods, namely Min, Max, Sum, and Average.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Aggregate<TSource,TAccumulate> (IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)

Source: Aggregate.cs

Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.

C#

```
public static TAccumulate Aggregate<TSource,TAccumulate> (this
System.Collections.Generic.IEnumerable<TSource> source, TAccumulate seed,
Func<TAccumulate,TSource,TAccumulate> func);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`func` Func<TAccumulate,TSource,TAccumulate>

An accumulator function to be invoked on each element.

Returns

TAccumulate

The final accumulator value.

Exceptions

ArgumentNullException

`source` or `func` is `null`.

Examples

The following code example demonstrates how to use Aggregate to apply an accumulator function and use a seed value.

C#

```
int[] ints = { 4, 8, 8, 3, 9, 0, 7, 8, 2 };

// Count the even numbers in the array, using a seed value of 0.
int numEven = ints.Aggregate(0, (total, next) =>
                                    next % 2 == 0 ? total + 1 : total);

Console.WriteLine("The number of even integers is: {0}", numEven);

// This code produces the following output:
//
// The number of even integers is: 6
```

Remarks

The Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>) method makes it simple to perform a calculation over a sequence of values. This method works by calling `func` one time for each element in `source`. Each time `func` is called, Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate,

Func<TAccumulate,TSource,TAccumulate>) passes both the element from the sequence and an aggregated value (as the first argument to `func`). The value of the `seed` parameter is used as the initial aggregate value. The result of `func` replaces the previous aggregated value. Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>) returns the final result of `func`.

To simplify common aggregation operations, the standard query operators also include a general purpose count method, Count, and four numeric aggregation methods, namely Min, Max, Sum, and Average.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)

Source: Aggregate.cs

Applies an accumulator function over a sequence.

C#

```
public static TSource Aggregate<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TSource,TSource> func);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to aggregate over.

`func` Func<TSource,TSource,TSource>

An accumulator function to be invoked on each element.

Returns

TSource

The final accumulator value.

Exceptions

ArgumentNullException

`source` or `func` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to reverse the order of words in a string by using Aggregate.

C#

```
string sentence = "the quick brown fox jumps over the lazy dog";

// Split the string into individual words.
string[] words = sentence.Split(' ');

// Prepend each word to the beginning of the
// new sentence to reverse the word order.
string reversed = words.Aggregate((workingSentence, next) =>
                                  next + " " + workingSentence);

Console.WriteLine(reversed);

// This code produces the following output:
```

```
//
// dog lazy the over jumps fox brown quick the
```

Remarks

The Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>) method makes it simple to perform a calculation over a sequence of values. This method works by calling `func` one time for each element in `source` except the first one. Each time `func` is called, Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>) passes both the element from the sequence and an aggregated value (as the first argument to `func`). The first element of `source` is used as the initial aggregate value. The result of `func` replaces the previous aggregated value. Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>) returns the final result of `func`.

This overload of the Aggregate method isn't suitable for all cases because it uses the first element of `source` as the initial aggregate value. You should choose another overload if the return value should include only the elements of `source` that meet a certain condition. For example, this overload isn't reliable if you want to calculate the sum of the even numbers in `source`. The result will be incorrect if the first element is odd instead of even.

To simplify common aggregation operations, the standard query operators also include a general purpose count method, Count, and four numeric aggregation methods, namely Min, Max, Sum, and Average.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.All<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: AnyAll.cs

Determines whether all elements of a sequence satisfy a condition.

```
C#

public static bool All<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,bool>
predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> that contains the elements to apply the predicate to.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Boolean

`true` if every element of the source sequence passes the test in the specified predicate, or if the sequence is empty; otherwise, `false`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use All to determine whether all the elements in a sequence satisfy a condition. Variable `allStartWithB` is true if all the pet names start with "B" or if the `pets` array is empty.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void AllEx()
{
    // Create an array of Pets.
    Pet[] pets = { new Pet { Name="Barley", Age=10 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=6 } };

    // Determine whether all pet names
    // in the array start with 'B'.
    bool allStartWithB = pets.All(pet =>
                                      pet.Name.StartsWith("B"));

    Console.WriteLine(
        "{0} pet names start with 'B'.",
        allStartWithB ? "All" : "Not all");
}

// This code produces the following output:
//
//  Not all pet names start with 'B'.
```

The Boolean value that the All method returns is typically used in the predicate of a `where` clause (`Where` clause in Visual Basic) or a direct call to the Where method. The following example demonstrates this use of the `All` method.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}
class Person
```

```
{
    public string LastName { get; set; }
    public Pet[] Pets { get; set; }
}

public static void AllEx2()
{
    List<Person> people = new List<Person>
        { new Person { LastName = "Haas",
                       Pets = new Pet[] { new Pet { Name="Barley", Age=10 },
                                          new Pet { Name="Boots", Age=14 },
                                          new Pet { Name="Whiskers", Age=6
}}},
          new Person { LastName = "Fakhouri",
                       Pets = new Pet[] { new Pet { Name = "Snowball", Age =
1}}},
          new Person { LastName = "Antebi",
                       Pets = new Pet[] { new Pet { Name = "Belle", Age = 8}
}},
          new Person { LastName = "Philips",
                       Pets = new Pet[] { new Pet { Name = "Sweetie", Age =
2},
                                          new Pet { Name = "Rover", Age =
13}} }
        };

    // Determine which people have pets that are all older than 5.
    IEnumerable<string> names = from person in people
                                where person.Pets.All(pet => pet.Age > 5)
                                select person.LastName;

    foreach (string name in names)
    {
        Console.WriteLine(name);
    }

    /* This code produces the following output:
     *
     * Haas
     * Antebi
     */
}
```

Remarks

> **Note**
>
> This method does not return all the elements of a collection. Instead, it determines whether all the elements of a collection satisfy a condition.

The enumeration of `source` is stopped as soon as the result can be determined.

In Visual Basic query expression syntax, an `Aggregate Into All()` clause translates to an invocation of All.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Aggregate Clause (Visual Basic)

Enumerable.Any Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Determines whether any element of a sequence exists or satisfies a condition.

Overloads

Expand table

Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.

Any<TSource>(IEnumerable<TSource>)

Source: AnyAll.cs

Determines whether a sequence contains any elements.

C#

```
public static bool Any<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

The IEnumerable<T> to check for emptiness.

Returns

Boolean

`true` if the source sequence contains any elements; otherwise, `false`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Any to determine whether a sequence contains any elements.

C#

```
List<int> numbers = new List<int> { 1, 2 };
bool hasElements = numbers.Any();

Console.WriteLine("The list {0} empty.",
    hasElements ? "is not" : "is");

// This code produces the following output:
//
// The list is not empty.
```

The Boolean value that the Any<TSource>(IEnumerable<TSource>) method returns is typically used in the predicate of a `where` clause (`Where` clause in Visual Basic) or a direct call to the Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>) method. The following example demonstrates this use of the `Any` method.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}
class Person
{
    public string LastName { get; set; }
    public Pet[] Pets { get; set; }
}

public static void AnyEx2()
{
```

```
    List<Person> people = new List<Person>
        { new Person { LastName = "Haas",
                       Pets = new Pet[] { new Pet { Name="Barley", Age=10
},
                                          new Pet { Name="Boots", Age=14
},
                                          new Pet { Name="Whiskers",
Age=6 }}},
          new Person { LastName = "Fakhouri",
                       Pets = new Pet[] { new Pet { Name = "Snowball",
Age = 1}}},
          new Person { LastName = "Antebi",
                       Pets = new Pet[] { }},
          new Person { LastName = "Philips",
                       Pets = new Pet[] { new Pet { Name = "Sweetie", Age
= 2},
                                          new Pet { Name = "Rover", Age =
13}} }
        };

    // Determine which people have a non-empty Pet array.
    IEnumerable<string> names = from person in people
                                where person.Pets.Any()
                                select person.LastName;

    foreach (string name in names)
    {
        Console.WriteLine(name);
    }

    /* This code produces the following output:

       Haas
       Fakhouri
       Philips
    */
}
```

Remarks

> **Note**
>
> This method does not return any one element of a collection. Instead, it determines whether the collection contains any elements.

The enumeration of `source` is stopped as soon as the result can be determined.

In Visual Basic query expression syntax, an `Aggregate Into Any()` clause translates to an invocation of Any.

See also

- Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)
- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: AnyAll.cs

Determines whether any element of a sequence satisfies a condition.

C#

```
public static bool Any<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to apply the predicate to.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Boolean

`true` if the source sequence is not empty and at least one of its elements passes the test in the specified predicate; otherwise, `false`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use Any to determine whether any element in a sequence satisfies a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
    public bool Vaccinated { get; set; }
}

public static void AnyEx3()
{
    // Create an array of Pets.
    Pet[] pets =
        { new Pet { Name="Barley", Age=8, Vaccinated=true },
          new Pet { Name="Boots", Age=4, Vaccinated=false },
          new Pet { Name="Whiskers", Age=1, Vaccinated=false } };

    // Determine whether any pets over age 1 are also unvaccinated.
    bool unvaccinated =
        pets.Any(p => p.Age > 1 && p.Vaccinated == false);

    Console.WriteLine(
        "There {0} unvaccinated animals over age one.",
        unvaccinated ? "are" : "are not any");
}

// This code produces the following output:
```

```
//
//  There are unvaccinated animals over age one.
```

Remarks

> **Note**
>
> This method does not return any one element of a collection. Instead, it determines whether any elements of a collection satisfy a condition.

The enumeration of `source` is stopped as soon as the result can be determined.

In Visual Basic query expression syntax, an `Aggregate Into Any()` clause translates to an invocation of Any.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Append<TSource> (IEnumerable<TSource>, TSource) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: AppendPrepend.cs

Appends a value to the end of the sequence.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Append<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, TSource element);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

A sequence of values.

element TSource

The value to append to `source`.

Returns

IEnumerable<TSource>

A new sequence that ends with `element`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Append to append a value to the end of the sequence.

C#

```
// Creating a list of numbers
List<int> numbers = new List<int> { 1, 2, 3, 4 };

// Trying to append any value of the same type
numbers.Append(5);

// It doesn't work because the original list has not been changed
Console.WriteLine(string.Join(", ", numbers));

// It works now because we are using a changed copy of the original list
Console.WriteLine(string.Join(", ", numbers.Append(5)));

// If you prefer, you can create a new list explicitly
List<int> newNumbers = numbers.Append(5).ToList();

// And then write to the console output
Console.WriteLine(string.Join(", ", newNumbers));

// This code produces the following output:
//
// 1, 2, 3, 4
// 1, 2, 3, 4, 5
// 1, 2, 3, 4, 5
```

Remarks

> **Note**
>
> This method does not modify the elements of the collection. Instead, it creates a copy of the collection with the new element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.6, 2.0, 2.1

Enumerable.AsEnumerable<TSource> (IEnumerable<TSource>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Enumerable.cs

Returns the input typed as IEnumerable<T>.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
AsEnumerable<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to type as IEnumerable<T>.

Returns

IEnumerable<TSource>

The input sequence typed as IEnumerable<T>.

Examples

The following code example demonstrates how to use AsEnumerable<TSource> (IEnumerable<TSource>) to hide a type's custom `Where` method when the standard query operator implementation is desired.

C#

```
// Custom class.
class Clump<T> : List<T>
{
    // Custom implementation of Where().
    public IEnumerable<T> Where(Func<T, bool> predicate)
    {
        Console.WriteLine("In Clump's implementation of Where().");
        return Enumerable.Where(this, predicate);
    }
}

static void AsEnumerableEx1()
{
    // Create a new Clump<T> object.
    Clump<string> fruitClump =
        new Clump<string> { "apple", "passionfruit", "banana",
            "mango", "orange", "blueberry", "grape", "strawberry" };

    // First call to Where():
    // Call Clump's Where() method with a predicate.
    IEnumerable<string> query1 =
        fruitClump.Where(fruit => fruit.Contains("o"));

    Console.WriteLine("query1 has been created.\n");

    // Second call to Where():
    // First call AsEnumerable() to hide Clump's Where() method and thereby
    // force System.Linq.Enumerable's Where() method to be called.
    IEnumerable<string> query2 =
        fruitClump.AsEnumerable().Where(fruit => fruit.Contains("o"));

    // Display the output.
    Console.WriteLine("query2 has been created.");
}

// This code produces the following output:
//
// In Clump's implementation of Where().
// query1 has been created.
//
// query2 has been created.
```

Remarks

The AsEnumerable<TSource>(IEnumerable<TSource>) method has no effect other than to change the compile-time type of `source` from a type that implements IEnumerable<T> to IEnumerable<T> itself.

AsEnumerable<TSource>(IEnumerable<TSource>) can be used to choose between query implementations when a sequence implements IEnumerable<T> but also has a different set of public query methods available. For example, given a generic class `Table` that implements IEnumerable<T> and has its own methods such as `Where`, `Select`, and `SelectMany`, a call to `Where` would invoke the public `Where` method of `Table`. A `Table` type that represents a database table could have a `Where` method that takes the predicate argument as an expression tree and converts the tree to SQL for remote execution. If remote execution is not desired, for example because the predicate invokes a local method, the AsEnumerable method can be used to hide the custom methods and instead make the standard query operators available.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Average Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Computes the average of a sequence of numeric values.

Overloads

Expand table

Average(IEnumerable<Single>)	Computes the average of a sequence of Single values.
Average(IEnumerable<Nullable<Single>>)	Computes the average of a sequence of nullable Single values.
Average(IEnumerable<Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values.
Average(IEnumerable<Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values.
Average(IEnumerable<Nullable<Double>>)	Computes the average of a sequence of nullable Double values.
Average(IEnumerable<Double>)	Computes the average of a sequence of Double values.
Average(IEnumerable<Int64>)	Computes the average of a sequence of Int64 values.
Average(IEnumerable<Int32>)	Computes the average of a sequence of Int32 values.
Average(IEnumerable<Decimal>)	Computes the average of a sequence of Decimal values.
Average(IEnumerable<Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.

Average(IEnumerable<Single>)

Source: Average.cs

Computes the average of a sequence of Single values.

```
C#

public static float Average (this
System.Collections.Generic.IEnumerable<float> source);
```

Parameters

`source` IEnumerable<Single>

A sequence of Single values to calculate the average of.

Returns

Single

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Nullable<Single>>)

Source: Average.cs

Computes the average of a sequence of nullable Single values.

C#

```
public static float? Average (this
System.Collections.Generic.IEnumerable<float?> source);
```

Parameters

`source` IEnumerable<Nullable<Single>>

A sequence of nullable Single values to calculate the average of.

Returns

Nullable<Single>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Nullable<Int64>>)

Source: Average.cs

Computes the average of a sequence of nullable Int64 values.

C#

```
public static double? Average (this
System.Collections.Generic.IEnumerable<long?> source);
```

Parameters

`source` IEnumerable<Nullable<Int64>>

A sequence of nullable Int64 values to calculate the average of.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum of the elements in the sequence is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Average(IEnumerable<Nullable<Int64>>) to calculate an average.

C#

```
long?[] longs = { null, 10007L, 37L, 399846234235L };

double? average = longs.Average();

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 133282081426.333.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Nullable<Int32>>)

Source: Average.cs

Computes the average of a sequence of nullable Int32 values.

C#

```
public static double? Average (this
System.Collections.Generic.IEnumerable<int?> source);
```

Parameters

`source` IEnumerable<Nullable<Int32>>

A sequence of nullable Int32 values to calculate the average of.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum of the elements in the sequence is larger than Int64.MaxValue.

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Nullable<Double>>)

Source: Average.cs

Computes the average of a sequence of nullable Double values.

C#

```
public static double? Average (this
System.Collections.Generic.IEnumerable<double?> source);
```

Parameters

source IEnumerable<Nullable<Double>>

A sequence of nullable Double values to calculate the average of.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

If the sum of the elements is too large to represent as a Double, this method returns positive or negative infinity.

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Double>)

Source: Average.cs

Computes the average of a sequence of Double values.

```
C#

public static double Average (this
System.Collections.Generic.IEnumerable<double> source);
```

Parameters

`source` IEnumerable<Double>

A sequence of Double values to calculate the average of.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

If the sum of the elements is too large to represent as a Double, this method returns positive or negative infinity.

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Int64>)

Source: Average.cs

Computes the average of a sequence of Int64 values.

C#

```
public static double Average (this
System.Collections.Generic.IEnumerable<long> source);
```

Parameters

`source` IEnumerable<Int64>

A sequence of Int64 values to calculate the average of.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Int32>)

Source: Average.cs

Computes the average of a sequence of Int32 values.

C#

```
public static double Average (this
System.Collections.Generic.IEnumerable<int> source);
```

Parameters

`source` IEnumerable<Int32>

A sequence of Int32 values to calculate the average of.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average(IEnumerable<Int32>) to calculate an average.

```C#
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };

double average = grades.Average();

Console.WriteLine("The average grade is {0}.", average);

// This code produces the following output:
//
// The average grade is 77.6.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Decimal>)

Source: Average.cs

Computes the average of a sequence of Decimal values.

C#

```
public static decimal Average (this
System.Collections.Generic.IEnumerable<decimal> source);
```

Parameters

`source` IEnumerable<Decimal>

A sequence of Decimal values to calculate the average of.

Returns

Decimal

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average(IEnumerable<Nullable<Decimal>>)

Source: Average.cs

Computes the average of a sequence of nullable Decimal values.

C#

```
public static decimal? Average (this
System.Collections.Generic.IEnumerable<decimal?> source);
```

Parameters

`source` IEnumerable<Nullable<Decimal>>

A sequence of nullable Decimal values to calculate the average of.

Returns

Nullable<Decimal>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum of the elements in the sequence is larger than Decimal.MaxValue.

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)

Source: Average.cs

Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.

```
C#
```

```
public static double? Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int?
> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum of the elements in the sequence is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to calculate an average.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the

overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

double average = fruits.Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)

Source: Average.cs

Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static float Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to calculate an average.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

double average = fruits.Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)

Source: Average.cs

Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double? Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Examples

The following code example demonstrates how to use Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>) to calculate an average.

> ⓘ **Note**

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

```C#
string[] numbers = { "10007", "37", "299846234235" };

double average = numbers.Average(num => long.Parse(num));

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 99948748093.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)

Source: Average.cs

Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double? Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int64>) to calculate an average.

Note

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] numbers = { "10007", "37", "299846234235" };

double average = numbers.Average(num => long.Parse(num));

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 99948748093.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)

Source: Average.cs

Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static float? Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to calculate an average.

Note

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

double average = fruits.Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)

Source: Average.cs

Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of the elements of source.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

OverflowException

The sum of the elements in the sequence is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int64>) to calculate an average.

C#

```
string[] numbers = { "10007", "37", "299846234235" };

double average = numbers.Average(num => long.Parse(num));

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 99948748093.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)

Source: Average.cs

Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

OverflowException

The sum of the elements in the sequence is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to calculate an average.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

double average = fruits.Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)

Source: Average.cs

Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int64>) to calculate an average.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] numbers = { "10007", "37", "299846234235" };

double average = numbers.Average(num => long.Parse(num));

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 99948748093.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)

Source: Average.cs

Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static decimal Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

OverflowException

The sum of the elements in the sequence is larger than Decimal.MaxValue.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int64>) to calculate an average.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] numbers = { "10007", "37", "299846234235" };

double average = numbers.Average(num => long.Parse(num));

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 99948748093.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)

Source: Average.cs

Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static decimal? Average<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to calculate the average of.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The average of the sequence of values, or `null` if the source sequence is empty or contains only values that are `null`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum of the elements in the sequence is larger than Decimal.MaxValue.

Examples

The following code example demonstrates how to use Average<TSource> (IEnumerable<TSource>, Func<TSource,Int64>) to calculate an average.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] numbers = { "10007", "37", "299846234235" };

double average = numbers.Average(num => long.Parse(num));

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 99948748093.
```

Remarks

In Visual Basic query expression syntax, an `Aggregate Into Average()` clause translates to an invocation of Average.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Cast<TResult> (IEnumerable) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Cast.cs

Casts the elements of an IEnumerable to the specified type.

C#

```
public static System.Collections.Generic.IEnumerable<TResult> Cast<TResult>
(this System.Collections.IEnumerable source);
```

Type Parameters

`TResult`

The type to cast the elements of `source` to.

Parameters

`source` IEnumerable

The IEnumerable that contains the elements to be cast to type `TResult`.

Returns

IEnumerable<TResult>

An IEnumerable<T> that contains each element of the source sequence cast to the specified type.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidCastException

An element in the sequence cannot be cast to type `TResult`.

Examples

The following code example demonstrates how to use Cast<TResult>(IEnumerable) to enable the use of the standard query operators on an ArrayList.

C#

```
System.Collections.ArrayList fruits = new System.Collections.ArrayList();
fruits.Add("mango");
fruits.Add("apple");
fruits.Add("lemon");

IEnumerable<string> query =
    fruits.Cast<string>().OrderBy(fruit => fruit).Select(fruit => fruit);

// The following code, without the cast, doesn't compile.
//IEnumerable<string> query1 =
//    fruits.OrderBy(fruit => fruit).Select(fruit => fruit);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}

// This code produces the following output:
//
// apple
// lemon
// mango
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The Cast<TResult>(IEnumerable) method enables the standard query operators to be invoked on non-generic collections by supplying the necessary type information. For example, ArrayList does not implement IEnumerable<T>, but by calling Cast<TResult>(IEnumerable) on the ArrayList object, the standard query operators can then be used to query the sequence.

If an element cannot be converted to type `TResult`, this method throws a InvalidCastException.

The source sequence for this method is IEnumerable, which means the elements have the compile-time static type of `object`. The only type conversions that are performed by this method are reference conversions and unboxing conversions. The runtime type of the elements in the collection must match the target type, or in the case of value types, the runtime type of elements must be the result of a boxing conversion of the target type. Other conversion types, such as those between different numeric types, are not allowed.

To obtain only those elements that can be converted to type `TResult`, use the OfType method instead of Cast<TResult>(IEnumerable).

In a query expression, an explicitly typed iteration variable translates to an invocation of Cast<TResult>(IEnumerable). This example shows the syntax for an explicitly typed range variable.

```
C#

from int i in objects
```

Use the `select` clause of a query to perform other conversion types, like the implicit numeric conversions. The following example uses both the `Cast` method and a `select` statement to convert a sequence of boxed integers to a sequence of doubles.

```
C#

IEnumerable sequence = Enumerable.Range(0, 10);
var doubles = from int item in sequence
                select (double)item;
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- from clause (C# Reference)
- From Clause (Visual Basic)

Enumerable.Chunk<TSource> (IEnumerable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Chunk.cs

Splits the elements of a sequence into chunks of size at most `size`.

C#

```
public static System.Collections.Generic.IEnumerable<TSource[]>
Chunk<TSource> (this System.Collections.Generic.IEnumerable<TSource> source,
int size);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to chunk.

`size` Int32

The maximum size of each chunk.

Returns

IEnumerable<TSource[]>

An IEnumerable<T> that contains the elements the input sequence split into chunks of size `size`.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentOutOfRangeException

`size` is below 1.

Remarks

Each chunk except the last one will be of size `size`. The last chunk will contain the remaining elements and may be of a smaller size.

Applies to

Product	Versions
.NET	6, 7, 8, 9

See also

- Partitioning data

Enumerable.Concat<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Concat.cs

Concatenates two sequences.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Concat<TSource> (this System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

The first sequence to concatenate.

`second` IEnumerable<TSource>

The sequence to concatenate to the first sequence.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the concatenated elements of the two input sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following code example demonstrates how to use Concat<TSource> (IEnumerable<TSource>, IEnumerable<TSource>) to concatenate two sequences.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

static Pet[] GetCats()
{
    Pet[] cats = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
    return cats;
}

static Pet[] GetDogs()
{
    Pet[] dogs = { new Pet { Name="Bounder", Age=3 },
                   new Pet { Name="Snoopy", Age=14 },
                   new Pet { Name="Fido", Age=9 } };
    return dogs;
}

public static void ConcatEx1()
{
    Pet[] cats = GetCats();
    Pet[] dogs = GetDogs();

    IEnumerable<string> query =
        cats.Select(cat => cat.Name).Concat(dogs.Select(dog => dog.Name));

    foreach (string name in query)
    {
        Console.WriteLine(name);
    }
}

// This code produces the following output:
//
// Barley
// Boots
// Whiskers
// Bounder
// Snoopy
// Fido
```

An alternative way of concatenating two sequences is to construct a collection, for example an array, of sequences and then apply the SelectMany method, passing it the identity selector function. The following example demonstrates this use of SelectMany.

C#

```
Pet[] cats = GetCats();
Pet[] dogs = GetDogs();

IEnumerable<string> query =
    new[] { cats.Select(cat => cat.Name), dogs.Select(dog => dog.Name) }
    .SelectMany(name => name);

foreach (string name in query)
{
    Console.WriteLine(name);
}

// This code produces the following output:
//
// Barley
// Boots
// Whiskers
// Bounder
// Snoopy
// Fido
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>) method differs from the Union method because the Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>) method returns all the original elements in the input sequences. The Union method returns only unique elements.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Contains Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Determines whether a sequence contains a specified element.

Overloads

Expand table

Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.

Contains<TSource>(IEnumerable<TSource>, TSource)

Source: Contains.cs

Determines whether a sequence contains a specified element by using the default equality comparer.

C#

```
public static bool Contains<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, TSource value);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence in which to locate a value.

`value` TSource

The value to locate in the sequence.

Returns

Boolean

`true` if the source sequence contains an element that has the specified value; otherwise, `false`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Contains<TSource> (IEnumerable<TSource>, TSource) to determine whether an array contains a specific element.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

string fruit = "mango";

bool hasMango = fruits.Contains(fruit);

Console.WriteLine(
    "The array {0} contain '{1}'.",
    hasMango ? "does" : "does not",
    fruit);

// This code produces the following output:
//
// The array does contain 'mango'.
```

Remarks

If the type of `source` implements ICollection<T>, the `Contains` method in that implementation is invoked to obtain the result. Otherwise, this method determines whether `source` contains the specified element.

Enumeration is terminated as soon as a matching element is found.

Elements are compared to the specified value by using the default equality comparer, Default.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)

Source: Contains.cs

Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.

C#

```
public static bool Contains<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, TSource value,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence in which to locate a value.

`value` TSource

The value to locate in the sequence.

`comparer` IEqualityComparer<TSource>

An equality comparer to compare values.

Returns

Boolean

`true` if the source sequence contains an element that has the specified value; otherwise, `false`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following example shows how to implement an equality comparer that can be used in the Contains method.

C#

```
public class Product
{
    public string Name { get; set; }
    public int Code { get; set; }
}

// Custom comparer for the Product class
class ProductComparer : IEqualityComparer<Product>
{
    // Products are equal if their names and product numbers are equal.
    public bool Equals(Product x, Product y)
    {

        //Check whether the compared objects reference the same data.
        if (Object.ReferenceEquals(x, y)) return true;
```

```
        //Check whether any of the compared objects is null.
        if (Object.ReferenceEquals(x, null) || Object.ReferenceEquals(y,
null))
            return false;

        //Check whether the products' properties are equal.
        return x.Code == y.Code && x.Name == y.Name;
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public int GetHashCode(Product product)
    {
        //Check whether the object is null
        if (Object.ReferenceEquals(product, null)) return 0;

        //Get hash code for the Name field if it is not null.
        int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = product.Code.GetHashCode();

        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this comparer, you can use a sequence of `Product` objects in the Contains method, as shown in the following example:

C#

```
Product[] fruits = { new Product { Name = "apple", Code = 9 },
                     new Product { Name = "orange", Code = 4 },
                     new Product { Name = "lemon", Code = 12 } };

Product apple = new Product { Name = "apple", Code = 9 };
Product kiwi = new Product { Name = "kiwi", Code = 8 };

ProductComparer prodc = new ProductComparer();

bool hasApple = fruits.Contains(apple, prodc);
bool hasKiwi = fruits.Contains(kiwi, prodc);

Console.WriteLine("Apple? " + hasApple);
Console.WriteLine("Kiwi? " + hasKiwi);

/*
    This code produces the following output:

    Apple? True
```

```
    Kiwi? False
*/
```

Remarks

Enumeration is terminated as soon as a matching element is found.

If `comparer` is `null`, the default equality comparer, Default, is used to compare elements to the specified value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Count Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the number of elements in a sequence.

Overloads

Expand table

Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.

Count<TSource>(IEnumerable<TSource>)

Source: Count.cs

Returns the number of elements in a sequence.

C#

```
public static int Count<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

A sequence that contains elements to be counted.

Returns

Int32

The number of elements in the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The number of elements in `source` is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Count<TSource> (IEnumerable<TSource>) to count the elements in an array.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

try
{
    int numberOfFruits = fruits.Count();
    Console.WriteLine(
        "There are {0} fruits in the collection.",
        numberOfFruits);
}
catch (OverflowException)
{
    Console.WriteLine("The count is too large to store as an Int32.");
    Console.WriteLine("Try using the LongCount() method instead.");
}

// This code produces the following output:
//
// There are 6 fruits in the collection.
```

Remarks

If the type of `source` implements ICollection<T>, that implementation is used to obtain the count of elements. Otherwise, this method determines the count.

Use the LongCount method when you expect and want to allow the result to be greater than MaxValue.

In Visual Basic query expression syntax, an `Aggregate Into Count()` clause translates to an invocation of Count.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Count.cs

Returns a number that represents how many elements in the specified sequence satisfy a condition.

C#

```
public static int Count<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence that contains elements to be tested and counted.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Int32

A number that represents how many elements in the sequence satisfy the condition in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

OverflowException

The number of elements in `source` is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Count<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to count the elements in an array that satisfy a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public bool Vaccinated { get; set; }
}

public static void CountEx2()
{
    Pet[] pets = { new Pet { Name="Barley", Vaccinated=true },
                   new Pet { Name="Boots", Vaccinated=false },
                   new Pet { Name="Whiskers", Vaccinated=false } };
```

```
    try
    {
        int numberUnvaccinated = pets.Count(p => p.Vaccinated == false);
        Console.WriteLine("There are {0} unvaccinated animals.",
numberUnvaccinated);
    }
    catch (OverflowException)
    {
        Console.WriteLine("The count is too large to store as an
Int32.");
        Console.WriteLine("Try using the LongCount() method instead.");
    }
}

// This code produces the following output:
//
// There are 2 unvaccinated animals.
```

Remarks

If the type of `source` implements ICollection<T>, that implementation is used to obtain the count of elements. Otherwise, this method determines the count.

You should use the LongCount method when you expect and want to allow the result to be greater than MaxValue.

In Visual Basic query expression syntax, an `Aggregate Into Count()` clause translates to an invocation of Count.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Enumerable.DefaultIfEmpty Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the elements of an IEnumerable<T>, or a default valued singleton collection if the sequence is empty.

Overloads

Expand table

DefaultIfEmpty<TSource> (IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource> (IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.

DefaultIfEmpty<TSource> (IEnumerable<TSource>)

Source: DefaultIfEmpty.cs

Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.

C#

```
public static System.Collections.Generic.IEnumerable<TSource?>
DefaultIfEmpty<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to return a default value for if it is empty.

Returns

IEnumerable<TSource>

An IEnumerable<T> object that contains the default value for the `TSource` type if `source` is empty; otherwise, `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code examples demonstrate how to use DefaultIfEmpty<TSource> (IEnumerable<TSource>) to provide a default value in case the source sequence is empty.

This example uses a non-empty sequence.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void DefaultIfEmptyEx1()
{
    List<Pet> pets =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 } };

    foreach (Pet pet in pets.DefaultIfEmpty())
    {
        Console.WriteLine(pet.Name);
    }
}
```

```
/*
 This code produces the following output:

 Barley
 Boots
 Whiskers
*/
```

This example uses an empty sequence.

```C#
List<int> numbers = new List<int>();

foreach (int number in numbers.DefaultIfEmpty())
{
    Console.WriteLine(number);
}

/*
 This code produces the following output:

 0
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The default value for reference and nullable types is `null`.

This method can be used to produce a left outer join when it is combined with the GroupJoin method.

See also

- Join operations (C#)
- Join operations (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

DefaultIfEmpty<TSource> (IEnumerable<TSource>, TSource)

Source: DefaultIfEmpty.cs

Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
DefaultIfEmpty<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to return the specified value for if it is empty.

`defaultValue` TSource

The value to return if the sequence is empty.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains `defaultValue` if `source` is empty; otherwise, `source`.

Examples

The following code example demonstrates how to use the DefaultIfEmpty<TSource> (IEnumerable<TSource>, TSource) method and specify a default value. The first sequence is not empty and the second sequence is empty.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void DefaultIfEmptyEx2()
{
    Pet defaultPet = new Pet { Name = "Default Pet", Age = 0 };

    List<Pet> pets1 =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 } };

    foreach (Pet pet in pets1.DefaultIfEmpty(defaultPet))
    {
        Console.WriteLine("Name: {0}", pet.Name);
    }

    List<Pet> pets2 = new List<Pet>();

    foreach (Pet pet in pets2.DefaultIfEmpty(defaultPet))
    {
        Console.WriteLine("\nName: {0}", pet.Name);
    }
}

/*
 This code produces the following output:

 Name: Barley
 Name: Boots
 Name: Whiskers

 Name: Default Pet
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

This method can be used to produce a left outer join when it is combined with the GroupJoin method.

See also

- Join operations (C#)
- Join operations (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Distinct Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns distinct elements from a sequence.

Overloads

Expand table

Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.

Remarks

The result sequence is unordered.

Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Distinct.cs

Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Distinct<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to remove duplicate elements from.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following example shows how to implement an equality comparer that can be used in the Distinct method.

C#

```
public class Product
{
    public string Name { get; set; }
    public int Code { get; set; }
}

// Custom comparer for the Product class
class ProductComparer : IEqualityComparer<Product>
{
    // Products are equal if their names and product numbers are equal.
    public bool Equals(Product x, Product y)
    {

        //Check whether the compared objects reference the same data.
        if (Object.ReferenceEquals(x, y)) return true;
```

```
        //Check whether any of the compared objects is null.
        if (Object.ReferenceEquals(x, null) || Object.ReferenceEquals(y,
null))
            return false;

        //Check whether the products' properties are equal.
        return x.Code == y.Code && x.Name == y.Name;
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public int GetHashCode(Product product)
    {
        //Check whether the object is null
        if (Object.ReferenceEquals(product, null)) return 0;

        //Get hash code for the Name field if it is not null.
        int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = product.Code.GetHashCode();

        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this comparer, you can use a sequence of `Product` objects in the Distinct method, as shown in the following example:

C#

```
Product[] products = { new Product { Name = "apple", Code = 9 },
                       new Product { Name = "orange", Code = 4 },
                       new Product { Name = "apple", Code = 9 },
                       new Product { Name = "lemon", Code = 12 } };

// Exclude duplicates.

IEnumerable<Product> noduplicates =
    products.Distinct(new ProductComparer());

foreach (var product in noduplicates)
    Console.WriteLine(product.Name + " " + product.Code);

/*
    This code produces the following output:
    apple 9
    orange 4
```

```
    lemon 12
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>) method returns an unordered sequence that contains no duplicate values. If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Distinct<TSource>(IEnumerable<TSource>)

Source: Distinct.cs

Returns distinct elements from a sequence by using the default equality comparer to compare values.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Distinct<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to remove duplicate elements from.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Distinct<TSource> (IEnumerable<TSource>) to return distinct elements from a sequence of integers.

C#

```
List<int> ages = new List<int> { 21, 46, 46, 55, 17, 21, 55, 55 };

IEnumerable<int> distinctAges = ages.Distinct();

Console.WriteLine("Distinct ages:");

foreach (int age in distinctAges)
{
    Console.WriteLine(age);
}

/*
 This code produces the following output:

 Distinct ages:
 21
 46
 55
```

```
 17
*/
```

If you want to return distinct elements from sequences of objects of some custom data type, you have to implement the IEquatable<T> generic interface in the class. The following code example shows how to implement this interface in a custom data type and provide GetHashCode and Equals methods.

```C#
public class MyProduct : IEquatable<MyProduct>
{
    public string Name { get; set; }
    public int Code { get; set; }

    public bool Equals(MyProduct other)
    {
        //Check whether the compared object is null.
        if (Object.ReferenceEquals(other, null)) return false;

        //Check whether the compared object references the same data.
        if (Object.ReferenceEquals(this, other)) return true;

        //Check whether the products' properties are equal.
        return Code.Equals(other.Code) && Name.Equals(other.Name);
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public override int GetHashCode()
    {

        //Get hash code for the Name field if it is not null.
        int hashProductName = Name == null ? 0 : Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = Code.GetHashCode();

        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this interface, you can use a sequence of `Product` objects in the Distinct<TSource>(IEnumerable<TSource>) method, as shown in the following example:

```C#
```

```
MyProduct[] products = { new MyProduct { Name = "apple", Code = 9 },
                         new MyProduct { Name = "orange", Code = 4 },
                         new MyProduct { Name = "apple", Code = 9 },
                         new MyProduct { Name = "lemon", Code = 12 } };

// Exclude duplicates.

IEnumerable<MyProduct> noduplicates =
    products.Distinct();

foreach (var product in noduplicates)
    Console.WriteLine(product.Name + " " + product.Code);

/*
    This code produces the following output:
    apple 9
    orange 4
    lemon 12
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The Distinct<TSource>(IEnumerable<TSource>) method returns an unordered sequence that contains no duplicate values. It uses the default equality comparer, Default, to compare values.

In Visual Basic query expression syntax, a `Distinct` clause translates to an invocation of Distinct.

The default equality comparer, Default, is used to compare values of the types that implement the IEquatable<T> generic interface. To compare a custom data type, you need to implement this interface and provide your own GetHashCode and Equals methods for the type.

For an example that uses IEqualityComparer<T> to define a custom comparer, see Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>).

See also

- DistinctBy
- Distinct Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.DistinctBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

DistinctBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.

DistinctBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)

Source: Distinct.cs

Returns distinct elements from a sequence according to a specified key selector function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
DistinctBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

`TKey`

The type of key to distinguish elements by.

Parameters

`source` IEnumerable<TSource>

The sequence to remove duplicate elements from.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) method returns an unordered sequence that contains no duplicate values. The default equality comparer, Default, is used to compare values.

See also

- Distinct and DistinctBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DistinctBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: Distinct.cs

Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
DistinctBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of key to distinguish elements by.

Parameters

`source` IEnumerable<TSource>

The sequence to remove duplicate elements from.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>) method returns an unordered sequence that contains no duplicate values. If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

See also

- Distinct and DistinctBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.ElementAt Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.

ElementAt<TSource>(IEnumerable<TSource>, Index)

Source: ElementAt.cs

Returns the element at a specified index in a sequence.

C#

```
public static TSource ElementAt<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Index index);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`index` Index

The index of the element to retrieve, which is either from the beginning or the end of the sequence.

Returns

TSource

The element at the specified position in the `source` sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentOutOfRangeException

`index` is outside the bounds of the `source` sequence.

Remarks

If the type of `source` implements IList<T>, that implementation is used to obtain the element at the specified index. Otherwise, this method obtains the specified element.

This method throws an exception if `index` is out of range. To instead return a default value when the specified index is out of range, use the ElementAtOrDefault method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ElementAt<TSource>(IEnumerable<TSource>, Int32)

Source: ElementAt.cs

Returns the element at a specified index in a sequence.

```
C#

public static TSource ElementAt<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, int index);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`index` Int32

The zero-based index of the element to retrieve.

Returns

TSource

The element at the specified position in the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentOutOfRangeException

`index` is less than 0 or greater than or equal to the number of elements in `source`.

Examples

The following code example demonstrates how to use ElementAt to return an element at a specific position.

```
C#

string[] names =
    { "Hartono, Tommy", "Adams, Terry", "Andersen, Henriette Thaulow",

        "Hedlund, Magnus", "Ito, Shu" };
```

```
Random random = new Random(DateTime.Now.Millisecond);

string name = names.ElementAt(random.Next(0, names.Length));

Console.WriteLine("The name chosen at random is '{0}'.", name);

/*
 This code produces output similar to the following:

 The name chosen at random is 'Ito, Shu'.
*/
```

Remarks

If the type of `source` implements IList<T>, that implementation is used to obtain the element at the specified index. Otherwise, this method obtains the specified element.

This method throws an exception if `index` is out of range. To instead return a default value when the specified index is out of range, use the ElementAtOrDefault method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ElementAtOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

ElementAtOrDefault<TSource> (IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource> (IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.

ElementAtOrDefault<TSource> (IEnumerable<TSource>, Index)

Source: ElementAt.cs

Returns the element at a specified index in a sequence or a default value if the index is out of range.

C#

```
public static TSource? ElementAtOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Index index);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`index` Index

The index of the element to retrieve, which is either from the beginning or the end of the sequence.

Returns

TSource

`default` if `index` is outside the bounds of the `source` sequence; otherwise, the element at the specified position in the `source` sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

If the type of `source` implements IList<T>, that implementation is used to obtain the element at the specified index. Otherwise, this method obtains the specified element.

The default value for reference and nullable types is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ElementAtOrDefault<TSource> (IEnumerable<TSource>, Int32)

Source: ElementAt.cs

Returns the element at a specified index in a sequence or a default value if the index is out of range.

```
C#

public static TSource? ElementAtOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, int index);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`index` Int32

The zero-based index of the element to retrieve.

Returns

TSource

`default`(`TSource`) if the index is outside the bounds of the source sequence; otherwise, the element at the specified position in the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use ElementAtOrDefault. This example uses an index that is outside the bounds of the array.

```
C#

string[] names =
    { "Hartono, Tommy", "Adams, Terry", "Andersen, Henriette Thaulow",
        "Hedlund, Magnus", "Ito, Shu" };
```

```
int index = 20;

string name = names.ElementAtOrDefault(index);

Console.WriteLine(
    "The name chosen at index {0} is '{1}'.",
    index,
    String.IsNullOrEmpty(name) ? "<no name at this index>" : name);

/*
 This code produces the following output:

 The name chosen at index 20 is '<no name at this index>'.
*/
```

Remarks

If the type of `source` implements IList<T>, that implementation is used to obtain the element at the specified index. Otherwise, this method obtains the specified element.

The default value for reference and nullable types is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Empty<TResult> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Enumerable.SpeedOpt.cs

Returns an empty IEnumerable<T> that has the specified type argument.

```
C#

public static System.Collections.Generic.IEnumerable<TResult> Empty<TResult>
();
```

Type Parameters

`TResult`

The type to assign to the type parameter of the returned generic IEnumerable<T>.

Returns

IEnumerable<TResult>

An empty IEnumerable<T> whose type argument is `TResult`.

Examples

The following code example demonstrates how to use Empty<TResult>() to generate an empty IEnumerable<T>.

```
C#

IEnumerable<decimal> empty = Enumerable.Empty<decimal>();
```

The following code example demonstrates a possible application of the Empty<TResult>() method. The Aggregate method is applied to a collection of string arrays. The elements of each array in the collection are added to the resulting IEnumerable<T> only if that array contains four or more elements. Empty is used to generate the seed value

for Aggregate because if no array in the collection has four or more elements, only the empty sequence is returned.

C#

```
string[] names1 = { "Hartono, Tommy" };
string[] names2 = { "Adams, Terry", "Andersen, Henriette Thaulow",
                    "Hedlund, Magnus", "Ito, Shu" };
string[] names3 = { "Solanki, Ajay", "Hoeing, Helge",
                    "Andersen, Henriette Thaulow",
                    "Potra, Cristina", "Iallo, Lucio" };

List<string[]> namesList =
    new List<string[]> { names1, names2, names3 };

// Only include arrays that have four or more elements
IEnumerable<string> allNames =
    namesList.Aggregate(Enumerable.Empty<string>(),
    (current, next) => next.Length > 3 ? current.Union(next) : current);

foreach (string name in allNames)
{
    Console.WriteLine(name);
}

/*
 This code produces the following output:

 Adams, Terry
 Andersen, Henriette Thaulow
 Hedlund, Magnus
 Ito, Shu
 Solanki, Ajay
 Hoeing, Helge
 Potra, Cristina
 Iallo, Lucio
*/
```

Remarks

The Empty<TResult>() method caches an empty sequence of type `TResult`. When the object it returns is enumerated, it yields no elements.

In some cases, this method is useful for passing an empty sequence to a user-defined method that takes an IEnumerable<T>. It can also be used to generate a neutral element for methods such as Union. See the Example section for an example of this use of Empty<TResult>().

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Except Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Produces the set difference of two sequences.

Overloads

Expand table

Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.

Remarks

The set difference of two sets is defined as the members of the first set that don't appear in the second set.

This method returns those elements in `first` that don't appear in `second`. It doesn't return those elements in `second` that don't appear in `first`. Only unique elements are returned.

Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)

Source: Except.cs

Produces the set difference of two sequences by using the default equality comparer to compare values.

```C#
public static System.Collections.Generic.IEnumerable<TSource>
Except<TSource> (this System.Collections.Generic.IEnumerable<TSource>
first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose elements that are not also in `second` will be returned.

`second` IEnumerable<TSource>

An IEnumerable<T> whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.

Returns

IEnumerable<TSource>

A sequence that contains the set difference of the elements of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following code example demonstrates how to use the `Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` method to compare two sequences of numbers and return elements that appear only in the first sequence.

```C#
double[] numbers1 = { 2.0, 2.0, 2.1, 2.2, 2.3, 2.3, 2.4, 2.5 };
double[] numbers2 = { 2.2 };

IEnumerable<double> onlyInFirstSet = numbers1.Except(numbers2);
```

```
foreach (double number in onlyInFirstSet)
    Console.WriteLine(number);

/*
 This code produces the following output:

 2
 2.1
 2.3
 2.4
 2.5
*/
```

If you want to compare sequences of objects of some custom data type, you have to implement the IEquatable<T> generic interface in a helper class. The following code example shows how to implement this interface in a custom data type and override the GetHashCode and Equals methods.

C#

```
public class ProductA : IEquatable<ProductA>
{
    public string Name { get; set; }
    public int Code { get; set; }

    public bool Equals(ProductA other)
    {
        if (other is null)
            return false;

        return this.Name == other.Name && this.Code == other.Code;
    }

    public override bool Equals(object obj) => Equals(obj as ProductA);
    public override int GetHashCode() => (Name, Code).GetHashCode();
}
```

After you implement this interface, you can use sequences of `ProductA` objects in the `Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` method, as shown in the following example:

C#

```
ProductA[] fruits1 = { new ProductA { Name = "apple", Code = 9 },
                       new ProductA { Name = "orange", Code = 4 },
                        new ProductA { Name = "lemon", Code = 12 } };

ProductA[] fruits2 = { new ProductA { Name = "apple", Code = 9 } };
```

```
// Get all the elements from the first array
// except for the elements from the second array.

IEnumerable<ProductA> except =
    fruits1.Except(fruits2);

foreach (var product in except)
    Console.WriteLine(product.Name + " " + product.Code);

/*
  This code produces the following output:

  orange 4
  lemon 12
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The default equality comparer, Default, is used to compare values of the types. To compare a custom data type, you need to override the Equals and the GetHashCode methods, and optionally implement the IEquatable<T> generic interface in the custom type. For more information, see the Default property.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Except.cs

Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Except<TSource> (this System.Collections.Generic.IEnumerable<TSource>
first, System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

first IEnumerable<TSource>

An IEnumerable<T> whose elements that are not also in `second` will be returned.

second IEnumerable<TSource>

An IEnumerable<T> whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.

comparer IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IEnumerable<TSource>

A sequence that contains the set difference of the elements of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

If you want to compare sequences of objects of some custom data type, you have to implement the IEqualityComparer<T> generic interface in a helper class. The following code example shows how to implement this interface in a custom data type and provide GetHashCode and Equals methods. The following example shows how to implement an equality comparer that can be used in the Except method.

C#

```
public class Product
{
    public string Name { get; set; }
    public int Code { get; set; }
}

// Custom comparer for the Product class
class ProductComparer : IEqualityComparer<Product>
{
    // Products are equal if their names and product numbers are equal.
    public bool Equals(Product x, Product y)
    {

        //Check whether the compared objects reference the same data.
        if (Object.ReferenceEquals(x, y)) return true;

        //Check whether any of the compared objects is null.
        if (Object.ReferenceEquals(x, null) || Object.ReferenceEquals(y,
null))
            return false;

        //Check whether the products' properties are equal.
        return x.Code == y.Code && x.Name == y.Name;
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public int GetHashCode(Product product)
    {
        //Check whether the object is null
        if (Object.ReferenceEquals(product, null)) return 0;

        //Get hash code for the Name field if it is not null.
        int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = product.Code.GetHashCode();
```

```
        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this comparer, you can use sequences of `Product` objects in the Except method, as shown in the following example:

```C#
Product[] fruits1 = { new Product { Name = "apple", Code = 9 },
                      new Product { Name = "orange", Code = 4 },
                       new Product { Name = "lemon", Code = 12 } };

Product[] fruits2 = { new Product { Name = "apple", Code = 9 } };

// Get all the elements from the first array
// except for the elements from the second array.

IEnumerable<Product> except =
    fruits1.Except(fruits2, new ProductComparer());

foreach (var product in except)
    Console.WriteLine(product.Name + " " + product.Code);

/*
  This code produces the following output:

  orange 4
  lemon 12
*/
```

Remarks

If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

See also

- ExceptBy

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ExceptBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.

ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)

Source: Except.cs

Produces the set difference of two sequences according to a specified key selector function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
ExceptBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TKey> second, Func<TSource,TKey>
keySelector);
```

Type Parameters

TSource

The type of the elements of the input sequence.

`TKey`

The type of key to identify elements by.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose keys that are not also in `second` will be returned.

`second` IEnumerable<TKey>

An IEnumerable<T> whose keys that also occur in the first sequence will cause those elements to be removed from the returned sequence.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

IEnumerable<TSource>

A sequence that contains the set difference of the elements of two sequences.

See also

- Except and ExceptBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ExceptBy<TSource,TKey> (IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: Except.cs

Produces the set difference of two sequences according to a specified key selector function.

```C#
public static System.Collections.Generic.IEnumerable<TSource>
ExceptBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TKey> second, Func<TSource,TKey>
keySelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequence.

`TKey`

The type of key to identify elements by.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose keys that are not also in `second` will be returned.

`second` IEnumerable<TKey>

An IEnumerable<T> whose keys that also occur in the first sequence will cause those elements to be removed from the returned sequence.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

The IEqualityComparer<T> to compare values.

Returns

IEnumerable<TSource>

A sequence that contains the set difference of the elements of two sequences.

See also

- Except and ExceptBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.First Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the first element of a sequence.

Overloads

Expand table

First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.

First<TSource>(IEnumerable<TSource>)

Source: First.cs

Returns the first element of a sequence.

C#

```
public static TSource First<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

The IEnumerable<T> to return the first element of.

Returns

TSource

The first element in the specified sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The source sequence is empty.

Examples

The following code example demonstrates how to use First<TSource> (IEnumerable<TSource>) to return the first element of an array.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                  83, 23, 87, 435, 67, 12, 19 };

int first = numbers.First();

Console.WriteLine(first);

/*
 This code produces the following output:

 9
*/
```

Remarks

The First<TSource>(IEnumerable<TSource>) method throws an exception if `source` contains no elements. To instead return a default value when the source sequence is empty, use the FirstOrDefault method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: First.cs

Returns the first element in a sequence that satisfies a specified condition.

C#

```
public static TSource First<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

The first element in the sequence that passes the test in the specified predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

No element satisfies the condition in `predicate`.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use First<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to return the first element of an array that satisfies a condition.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                    83, 23, 87, 435, 67, 12, 19 };

int first = numbers.First(number => number > 80);

Console.WriteLine(first);

/*
 This code produces the following output:

 92
*/
```

Remarks

The First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>) method throws an exception if no matching element is found in `source`. To instead return a default value when no matching element is found, use the FirstOrDefault method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.FirstOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the first element of a sequence, or a default value if no element is found.

Overloads

Expand table

FirstOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
FirstOrDefault<TSource> (IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource> (IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.

FirstOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>, TSource)

Source: First.cs

Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.

```
C#
```

```
public static TSource FirstOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if `source` is empty or if no element passes the test specified by `predicate`; otherwise, the first element in `source` that passes the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

FirstOrDefault<TSource> (IEnumerable<TSource>, TSource)

Source: First.cs

Returns the first element of a sequence, or a specified default value if the sequence contains no elements.

```
C#

public static TSource FirstOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The IEnumerable<T> to return the first element of.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if `source` is empty; otherwise, the first element in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

FirstOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)

Source: First.cs

Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.

```
C#

public static TSource? FirstOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

`default`(`TSource`) if `source` is empty or if no element passes the test specified by `predicate`; otherwise, the first element in `source` that passes the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use FirstOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) by passing in a predicate. In the second call to the method, there is no element in the array that satisfies the condition.

C#

```
string[] names = { "Hartono, Tommy", "Adams, Terry",
                     "Andersen, Henriette Thaulow",
                     "Hedlund, Magnus", "Ito, Shu" };

string firstLongName = names.FirstOrDefault(name => name.Length > 20);

Console.WriteLine("The first long name is '{0}'.", firstLongName);

string firstVeryLongName = names.FirstOrDefault(name => name.Length >
30);

Console.WriteLine(
    "There is {0} name longer than 30 characters.",
    string.IsNullOrEmpty(firstVeryLongName) ? "not a" : "a");

/*
 This code produces the following output:

 The first long name is 'Andersen, Henriette Thaulow'.
 There is not a name longer than 30 characters.
*/
```

Remarks

The default value for reference and nullable types is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

FirstOrDefault<TSource> (IEnumerable<TSource>)

Source: First.cs

Returns the first element of a sequence, or a default value if the sequence contains no elements.

```
C#

public static TSource? FirstOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The IEnumerable<T> to return the first element of.

Returns

TSource

`default`(`TSource`) if `source` is empty; otherwise, the first element in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use FirstOrDefault<TSource>(IEnumerable<TSource>) on an empty array.

C#

```
int[] numbers = { };
int first = numbers.FirstOrDefault();
Console.WriteLine(first);

/*
 This code produces the following output:

 0
*/
```

Sometimes the value of `default(TSource)` is not the default value that you want to use if the collection contains no elements. Instead of checking the result for the unwanted default value and then changing it if necessary, you can use the DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) method to specify the default value that you want to use if the collection is empty. Then, call First<TSource>(IEnumerable<TSource>) to obtain the first element. The following code example uses both techniques to obtain a default value of 1 if a collection of numeric months is empty. Because the default value for an integer is 0, which does not correspond to any month, the default value must be specified as 1 instead. The first result variable is checked for the unwanted default value after the query has finished executing. The second result variable is obtained by using DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) to specify a default value of 1.

C#

```
List<int> months = new List<int> { };

// Setting the default value to 1 after the query.
int firstMonth1 = months.FirstOrDefault();
if (firstMonth1 == 0)
{
    firstMonth1 = 1;
}
Console.WriteLine("The value of the firstMonth1 variable is {0}",
firstMonth1);
```

```
// Setting the default value to 1 by using DefaultIfEmpty() in the query.
int firstMonth2 = months.DefaultIfEmpty(1).First();
Console.WriteLine("The value of the firstMonth2 variable is {0}",
firstMonth2);

/*
 This code produces the following output:

 The value of the firstMonth1 variable is 1
 The value of the firstMonth2 variable is 1
*/
```

Remarks

The default value for reference and nullable types is `null`.

The FirstOrDefault method does not provide a way to specify a default value. If you want to specify a default value other than `default(TSource)`, use the DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) method as described in the Example section.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.GroupBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Groups the elements of a sequence.

Overloads

Expand table

GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are

	projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
GroupBy<TSource,TKey,TElement,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector,
Func<TKey,System.Collections.Generic.IEnumerable<TElement>,TResult>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in each IGrouping<TKey,TElement>.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

`resultSelector` Func<TKey,IEnumerable<TElement>,TResult>

A function to create a result value from each group.

Returns

IEnumerable<TResult>

A collection of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>) to group the projected elements of a sequence and then project a sequence of results of type `TResult`.

C#

```
class Pet
{
    public string Name { get; set; }
    public double Age { get; set; }
}

public static void GroupByEx4()
{
    // Create a list of pets.
    List<Pet> petsList =
        new List<Pet>{ new Pet { Name="Barley", Age=8.3 },
                       new Pet { Name="Boots", Age=4.9 },
                       new Pet { Name="Whiskers", Age=1.5 },
                       new Pet { Name="Daisy", Age=4.3 } };

    // Group Pet.Age values by the Math.Floor of the age.
    // Then project an anonymous type from each group
    // that consists of the key, the count of the group's
    // elements, and the minimum and maximum age in the group.
    var query = petsList.GroupBy(
        pet => Math.Floor(pet.Age),
        pet => pet.Age,
        (baseAge, ages) => new
        {
            Key = baseAge,
            Count = ages.Count(),
            Min = ages.Min(),
            Max = ages.Max()
        });

    // Iterate over each anonymous type.
    foreach (var result in query)
    {
        Console.WriteLine("\nAge group: " + result.Key);
        Console.WriteLine("Number of pets in this age group: " +
result.Count);
        Console.WriteLine("Minimum age: " + result.Min);
        Console.WriteLine("Maximum age: " + result.Max);
    }

    /*  This code produces the following output:
```

```
        Age group: 8
        Number of pets in this age group: 1
        Minimum age: 8.3
        Maximum age: 8.3

        Age group: 4
        Number of pets in this age group: 2
        Minimum age: 4.3
        Maximum age: 4.9

        Age group: 1
        Number of pets in this age group: 1
        Minimum age: 1.5
        Maximum age: 1.5
    */
}
```

Remarks

In query expression syntax, a `group by` (C#) or `Group By Into` (Visual Basic) clause translates to an invocation of GroupBy.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,

TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.

```
C#

public static System.Collections.Generic.IEnumerable<TResult>
GroupBy<TSource,TKey,TElement,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector,
Func<TKey,System.Collections.Generic.IEnumerable<TElement>,TResult>
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in each IGrouping<TKey,TElement>.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

`resultSelector` Func<TKey,IEnumerable<TElement>,TResult>

A function to create a result value from each group.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys with.

Returns

IEnumerable<TResult>

A collection of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` or `resultSelector` is `null`.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

GroupBy<TSource,TKey,TElement> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

C#

```
public static
System.Collections.Generic.IEnumerable<System.Linq.IGrouping<TKey,TElemen
t>> GroupBy<TSource,TKey,TElement> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in the IGrouping<TKey,TElement>.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

elementSelector Func<TSource,TElement>

A function to map each source element to an element in the IGrouping<TKey,TElement>.

Returns

IEnumerable<IGrouping<TKey,TElement>>

An `IEnumerable<IGrouping<TKey, TElement>>` in C# or `IEnumerable(Of IGrouping(Of TKey, TElement))` in Visual Basic where each IGrouping<TKey,TElement> object contains a collection of objects of type `TElement` and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>) to group the elements of a sequence.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

// Uses method-based query syntax.
public static void GroupByEx1()
{
    // Create a list of pets.
    List<Pet> pets =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 },
                       new Pet { Name="Daisy", Age=4 } };

    // Group the pets using Age as the key value
    // and selecting only the pet's Name for each value.
    IEnumerable<IGrouping<int, string>> query =
        pets.GroupBy(pet => pet.Age, pet => pet.Name);

    // Iterate over each IGrouping in the collection.
```

```
    foreach (IGrouping<int, string> petGroup in query)
    {
        // Print the key value of the IGrouping.
        Console.WriteLine(petGroup.Key);
        // Iterate over each value in the
        // IGrouping and print the value.
        foreach (string name in petGroup)
            Console.WriteLine("  {0}", name);
    }
}

/*
 This code produces the following output:

 8
   Barley
 4
   Boots
   Daisy
 1
   Whiskers
*/
```

In query expression syntax, a `group by` (C#) or `Group By Into` (Visual Basic) clause translates to an invocation of GroupBy. The translation of the query expression in the following example is equivalent to the query in the example above.

C#

```
IEnumerable<IGrouping<int, string>> query =
    from pet in pets
    group pet.Name by pet.Age;
```

> **Note**
>
> In a C# or Visual Basic query expression, the element and key selection expressions occur in the reverse order from their argument positions in a call to the GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>) method.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is

enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>) method returns a collection of IGrouping<TKey,TElement> objects, one for each distinct key that was encountered. An IGrouping<TKey,TElement> is an IEnumerable<T> that also has a key associated with its elements.

The IGrouping<TKey,TElement> objects are yielded in an order based on the order of the elements in `source` that produced the first key of each IGrouping<TKey,TElement>. Elements in a grouping are yielded in the order that the elements that produced them appear in `source`.

The default equality comparer Default is used to compare keys.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)

Source: Grouping.cs

Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

```C#
public static
System.Collections.Generic.IEnumerable<System.Linq.IGrouping<TKey,TElemen
t>> GroupBy<TSource,TKey,TElement> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in the IGrouping<TKey,TElement>.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IEnumerable<IGrouping<TKey,TElement>>

An `IEnumerable<IGrouping<TKey, TElement>>` in C# or `IEnumerable(Of IGrouping(Of TKey, TElement))` in Visual Basic where each IGrouping<TKey,TElement> object contains a collection of objects of type `TElement` and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

> **Note**
>
> For examples of `GroupBy`, see the following articles:
>
> - GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)
> - GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)
> - GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)

The GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>) method returns a collection of IGrouping<TKey,TElement> objects, one for each distinct key that was encountered. An IGrouping<TKey,TElement> is an IEnumerable<T> that also has a key associated with its elements.

The IGrouping<TKey,TElement> objects are yielded in an order based on the order of the elements in `source` that produced the first key of each

IGrouping<TKey,TElement>. Elements in a grouping are yielded in the order that the elements that produced them appear in `source`.

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

If two keys are considered equal according to `comparer`, the first key is chosen as the key for that grouping.

In query expression syntax, a `group by` (C#) or `Group By Into` (Visual Basic) clause translates to an invocation of GroupBy. For more information and usage examples, see group clause and Group By Clause.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

```
C#
```

```
public static System.Collections.Generic.IEnumerable<TResult>
GroupBy<TSource,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
Func<TKey,System.Collections.Generic.IEnumerable<TSource>,TResult>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`resultSelector` Func<TKey,IEnumerable<TSource>,TResult>

A function to create a result value from each group.

Returns

IEnumerable<TResult>

A collection of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>) to group the elements of a sequence and project a sequence of results of type `TResult`.

C#

```
class Pet
{
    public string Name { get; set; }
    public double Age { get; set; }
}

public static void GroupByEx3()
{
    // Create a list of pets.
    List<Pet> petsList =
        new List<Pet>{ new Pet { Name="Barley", Age=8.3 },
                       new Pet { Name="Boots", Age=4.9 },
                       new Pet { Name="Whiskers", Age=1.5 },
                       new Pet { Name="Daisy", Age=4.3 } };

    // Group Pet objects by the Math.Floor of their age.
    // Then project an anonymous type from each group
    // that consists of the key, the count of the group's
    // elements, and the minimum and maximum age in the group.
    var query = petsList.GroupBy(
        pet => Math.Floor(pet.Age),
        (age, pets) => new
        {
            Key = age,
            Count = pets.Count(),
            Min = pets.Min(pet => pet.Age),
            Max = pets.Max(pet => pet.Age)
        });

    // Iterate over each anonymous type.
    foreach (var result in query)
    {
        Console.WriteLine("\nAge group: " + result.Key);
        Console.WriteLine("Number of pets in this age group: " +
result.Count);
        Console.WriteLine("Minimum age: " + result.Min);
        Console.WriteLine("Maximum age: " + result.Max);
    }

    /*  This code produces the following output:

        Age group: 8
        Number of pets in this age group: 1
```

```
        Minimum age: 8.3
        Maximum age: 8.3

        Age group: 4
        Number of pets in this age group: 2
        Minimum age: 4.3
        Maximum age: 4.9

        Age group: 1
        Number of pets in this age group: 1
        Minimum age: 1.5
        Maximum age: 1.5
    */
}
```

Remarks

In query expression syntax, a `group by` (C#) or `Group By Into` (Visual Basic) clause translates to an invocation of GroupBy.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TResult> (IEnumerable<TSource>, Func<TSource,TKey>,

Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
GroupBy<TSource,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
Func<TKey,System.Collections.Generic.IEnumerable<TSource>,TResult>
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`resultSelector` Func<TKey,IEnumerable<TSource>,TResult>

A function to create a result value from each group.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys with.

Returns

IEnumerable<TResult>

A collection of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `resultSelector` is `null`.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function.

C#

```
public static
System.Collections.Generic.IEnumerable<System.Linq.IGrouping<TKey,TSource
>> GroupBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

IEnumerable<IGrouping<TKey,TSource>>

An `IEnumerable<IGrouping<TKey, TSource>>` in C# or `IEnumerable(Of IGrouping(Of TKey, TSource))` in Visual Basic where each IGrouping<TKey,TElement> object contains a sequence of objects and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is

enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

> **Note**
>
> For examples of `GroupBy`, see the following articles:
>
> - GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)
> - GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)
> - GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)

The GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) method returns a collection of IGrouping<TKey,TElement> objects, one for each distinct key that was encountered. An IGrouping<TKey,TElement> is an IEnumerable<T> that also has a key associated with its elements.

The IGrouping<TKey,TElement> objects are yielded in an order based on the order of the elements in `source` that produced the first key of each IGrouping<TKey,TElement>. Elements in a grouping are yielded in the order they appear in `source`.

The default equality comparer Default is used to compare keys.

In query expression syntax, a `group by` (C#) or `Group By Into` (Visual Basic) clause translates to an invocation of GroupBy. For more information and usage examples, see group clause and Group By Clause.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: Grouping.cs

Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

C#

```
public static
System.Collections.Generic.IEnumerable<System.Linq.IGrouping<TKey,TSource
>> GroupBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> whose elements to group.

keySelector Func<TSource,TKey>

A function to extract the key for each element.

comparer IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IEnumerable<IGrouping<TKey,TSource>>

An `IEnumerable<IGrouping<TKey, TSource>>` in C# or `IEnumerable(Of IGrouping(Of TKey, TSource))` in Visual Basic where each IGrouping<TKey,TElement> object contains a collection of objects and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

> **Note**
>
> For examples of `GroupBy`, see the following articles:
>
> - GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)
> - GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)
> - GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)

The GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>) method returns a collection of IGrouping<TKey,TElement> objects, one for each distinct key that was encountered. An IGrouping<TKey,TElement> is an IEnumerable<T> that also has a key associated with its elements.

The IGrouping<TKey,TElement> objects are yielded in an order based on the order of the elements in `source` that produced the first key of each IGrouping<TKey,TElement>. Elements in a grouping are yielded in the order they appear in `source`.

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

If two keys are considered equal according to `comparer`, the first key is chosen as the key for that grouping.

In query expression syntax, a `group by` (C#) or `Group By Into` (Visual Basic) clause translates to an invocation of GroupBy. For more information and usage examples, see group clause and Group By Clause.

See also

- group clause (C# Reference)
- Group By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.GroupJoin Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Correlates the elements of two sequences based on key equality, and groups the results.

Overloads

Expand table

GroupJoin<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.

GroupJoin<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)

Source: GroupJoin.cs

Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
```

```
System.Collections.Generic.IEnumerable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
Func<TOuter,System.Collections.Generic.IEnumerable<TInner>,TResult>
resultSelector);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IEnumerable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,IEnumerable<TInner>,TResult>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

Returns

IEnumerable<TResult>

An IEnumerable<T> that contains elements of type `TResult` that are obtained by performing a grouped join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>) to perform a grouped join on two sequences.

C#

```
class Person
{
    public string Name { get; set; }
}

class Pet
{
    public string Name { get; set; }
    public Person Owner { get; set; }
}

public static void GroupJoinEx1()
{
    Person magnus = new Person { Name = "Hedlund, Magnus" };
    Person terry = new Person { Name = "Adams, Terry" };
    Person charlotte = new Person { Name = "Weiss, Charlotte" };

    Pet barley = new Pet { Name = "Barley", Owner = terry };
    Pet boots = new Pet { Name = "Boots", Owner = terry };
    Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };
    Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

    List<Person> people = new List<Person> { magnus, terry, charlotte };
    List<Pet> pets = new List<Pet> { barley, boots, whiskers, daisy };

    // Create a list where each element is an anonymous
    // type that contains a person's name and
    // a collection of names of the pets they own.
    var query =
        people.GroupJoin(pets,
                         person => person,
```

```
                            pet => pet.Owner,
                            (person, petCollection) =>
                                new
                                {
                                    OwnerName = person.Name,
                                    Pets = petCollection.Select(pet =>
pet.Name)
                                });

    foreach (var obj in query)
    {
        // Output the owner's name.
        Console.WriteLine("{0}:", obj.OwnerName);
        // Output each of the owner's pet's names.
        foreach (string pet in obj.Pets)
        {
            Console.WriteLine("  {0}", pet);
        }
    }
}

/*
 This code produces the following output:

 Hedlund, Magnus:
   Daisy
 Adams, Terry:
   Barley
   Boots
 Weiss, Charlotte:
   Whiskers
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The default equality comparer, Default, is used to hash and compare keys.

GroupJoin produces hierarchical results, which means that elements from `outer` are paired with collections of matching elements from `inner`. `GroupJoin` enables you to base your results on a whole set of matches for each element of `outer`.

> ⓘ **Note**

If there are no correlated elements in `inner` for a given element of `outer`, the sequence of matches for that element will be empty but will still appear in the results.

The `resultSelector` function is called only one time for each `outer` element together with a collection of all the `inner` elements that match the `outer` element. This differs from the Join method, in which the result selector function is invoked on pairs that contain one element from `outer` and one element from `inner`.

`GroupJoin` preserves the order of the elements of `outer`, and for each element of `outer`, the order of the matching elements from `inner`.

GroupJoin has no direct equivalent in traditional relational database terms. However, this method does implement a superset of inner joins and left outer joins. Both of these operations can be written in terms of a grouped join. For more information, see Join operations.

In query expression syntax, a `join ... into` (C#) or `Group Join` (Visual Basic) clause translates to an invocation of GroupJoin.

See also

- Join operations (C#)
- Join operations (Visual Basic)
- join clause (C# Reference)
- Group Join Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

GroupJoin<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)

Source: GroupJoin.cs

Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
Func<TOuter,System.Collections.Generic.IEnumerable<TInner>,TResult>
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IEnumerable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,IEnumerable<TInner>,TResult>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to hash and compare keys.

Returns

IEnumerable<TResult>

An IEnumerable<T> that contains elements of type `TResult` that are obtained by performing a grouped join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

If `comparer` is `null`, the default equality comparer, Default, is used to hash and compare keys.

GroupJoin produces hierarchical results, which means that elements from `outer` are paired with collections of matching elements from `inner`. `GroupJoin` enables you to

base your results on a whole set of matches for each element of `outer`.

> **Note**
>
> If there are no correlated elements in `inner` for a given element of `outer`, the sequence of matches for that element will be empty but will still appear in the results.

The `resultSelector` function is called only one time for each `outer` element together with a collection of all the `inner` elements that match the `outer` element. This differs from the Join method in which the result selector function is invoked on pairs that contain one element from `outer` and one element from `inner`.

`GroupJoin` preserves the order of the elements of `outer`, and for each element of `outer`, the order of the matching elements from `inner`.

GroupJoin has no direct equivalent in traditional relational database terms. However, this method does implement a superset of inner joins and left outer joins. Both of these operations can be written in terms of a grouped join. For more information, see Join operations.

See also

- Join operations (C#)
- Join operations (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Intersect Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Produces the set intersection of two sequences.

Overloads

Expand table

Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)

Source: Intersect.cs

Produces the set intersection of two sequences by using the default equality comparer to compare values.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Intersect<TSource> (this System.Collections.Generic.IEnumerable<TSource>
first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in `second` will be returned.

`second` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in the first sequence will be returned.

Returns

IEnumerable<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following code example demonstrates how to use `Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` to return the elements that appear in each of two sequences of integers.

C#

```
int[] id1 = { 44, 26, 92, 30, 71, 38 };
int[] id2 = { 39, 59, 83, 47, 26, 4, 30 };

IEnumerable<int> both = id1.Intersect(id2);

foreach (int id in both)
    Console.WriteLine(id);

/*
 This code produces the following output:

 26
 30
*/
```

If you want to compare sequences of objects of some custom data type, you have to implement the IEquatable<T> generic interface in a helper class. The following code example shows how to implement this interface in a custom data type and override GetHashCode and Equals methods.

C#

```
public class ProductA : IEquatable<ProductA>
{
    public string Name { get; set; }
    public int Code { get; set; }

    public bool Equals(ProductA other)
    {
        if (other is null)
            return false;

        return this.Name == other.Name && this.Code == other.Code;
    }

    public override bool Equals(object obj) => Equals(obj as ProductA);
    public override int GetHashCode() => (Name, Code).GetHashCode();
}
```

After you implement this interface, you can use sequences of `ProductA` objects in the `Intersect` method, as shown in the following example:

C#

```
ProductA[] store1 = { new ProductA { Name = "apple", Code = 9 },
                      new ProductA { Name = "orange", Code = 4 } };

ProductA[] store2 = { new ProductA { Name = "apple", Code = 9 },
                      new ProductA { Name = "lemon", Code = 12 } };
```

C#

```
// Get the products from the first array
// that have duplicates in the second array.

IEnumerable<ProductA> duplicates =
    store1.Intersect(store2);

foreach (var product in duplicates)
    Console.WriteLine(product.Name + " " + product.Code);

/*
    This code produces the following output:
```

```
    apple 9
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The intersection of two sets A and B is defined as the set that contains all the elements of A that also appear in B, but no other elements.

When the object returned by this method is enumerated, `Intersect` yields distinct elements occurring in both sequences in the order in which they appear in `first`.

The default equality comparer, Default, is used to compare values of the types. To compare a custom data type, you need to override the Equals and the GetHashCode methods, and optionally implement the IEquatable<T> generic interface in the custom type. For more information, see the Default property.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Intersect.cs

Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Intersect<TSource> (this System.Collections.Generic.IEnumerable<TSource>
first, System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in `second` will be returned.

`second` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in the first sequence will be returned.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IEnumerable<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following example shows how to implement an equality comparer that can be used in the `Intersect` method.

C#

```
public class Product
{
    public string Name { get; set; }
    public int Code { get; set; }
}

// Custom comparer for the Product class
class ProductComparer : IEqualityComparer<Product>
{
    // Products are equal if their names and product numbers are equal.
    public bool Equals(Product x, Product y)
    {

        //Check whether the compared objects reference the same data.
        if (Object.ReferenceEquals(x, y)) return true;

        //Check whether any of the compared objects is null.
        if (Object.ReferenceEquals(x, null) || Object.ReferenceEquals(y,
null))
            return false;

        //Check whether the products' properties are equal.
        return x.Code == y.Code && x.Name == y.Name;
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public int GetHashCode(Product product)
    {
        //Check whether the object is null
        if (Object.ReferenceEquals(product, null)) return 0;

        //Get hash code for the Name field if it is not null.
        int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = product.Code.GetHashCode();

        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this comparer, you can use sequences of `Product` objects in the `Intersect` method, as shown in the following example:

```C#
Product[] store1 = { new Product { Name = "apple", Code = 9 },
                     new Product { Name = "orange", Code = 4 } };

Product[] store2 = { new Product { Name = "apple", Code = 9 },
                     new Product { Name = "lemon", Code = 12 } };

// Get the products from the first array
// that have duplicates in the second array.

IEnumerable<Product> duplicates =
    store1.Intersect(store2, new ProductComparer());

foreach (var product in duplicates)
    Console.WriteLine(product.Name + " " + product.Code);

/*
    This code produces the following output:
    apple 9
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The intersection of two sets A and B is defined as the set that contains all the elements of A that also appear in B, but no other elements.

When the object returned by this method is enumerated, `Intersect` yields distinct elements occurring in both sequences in the order in which they appear in `first`.

If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

See also

- IntersectBy

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.IntersectBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.

IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: Intersect.cs

Produces the set intersection of two sequences according to a specified key selector function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
IntersectBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TKey> second, Func<TSource,TKey>
keySelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in `second` will be returned.

`second` IEnumerable<TKey>

An IEnumerable<T> whose distinct elements that also appear in the first sequence will be returned.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IEnumerable<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is

enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The intersection of two sets A and B is defined as the set that contains all the elements of A that also appear in B, but no other elements.

When the object returned by this method is enumerated, `Intersect` yields distinct elements occurring in both sequences in the order in which they appear in `first`.

If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

See also

- Intersect and IntersectBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

IntersectBy<TSource,TKey> (IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)

Source: Intersect.cs

Produces the set intersection of two sequences according to a specified key selector function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
IntersectBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TKey> second, Func<TSource,TKey>
keySelector);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in `second` will be returned.

`second` IEnumerable<TKey>

An IEnumerable<T> whose distinct elements that also appear in the first sequence will be returned.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

IEnumerable<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The intersection of two sets A and B is defined as the set that contains all the elements of A that also appear in B, but no other elements.

When the object returned by this method is enumerated, `Intersect` yields distinct elements occurring in both sequences in the order in which they appear in `first`.

The default equality comparer, Default, is used to compare values.

See also

- Intersect and IntersectBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.Join Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Correlates the elements of two sequences based on matching keys.

Overloads

Expand table

Join<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

Join<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)

Source: Join.cs

Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Join<TOuter,TInner,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TOuter> outer,
```

```
System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
Func<TOuter,TInner,TResult> resultSelector);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IEnumerable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,TInner,TResult>

A function to create a result element from two matching elements.

Returns

IEnumerable<TResult>

An IEnumerable<T> that has elements of type `TResult` that are obtained by performing an inner join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>) to perform an inner join of two sequences based on a common key.

C#

```
class Person
{
    public string Name { get; set; }
}

class Pet
{
    public string Name { get; set; }
    public Person Owner { get; set; }
}

public static void JoinEx1()
{
    Person magnus = new Person { Name = "Hedlund, Magnus" };
    Person terry = new Person { Name = "Adams, Terry" };
    Person charlotte = new Person { Name = "Weiss, Charlotte" };

    Pet barley = new Pet { Name = "Barley", Owner = terry };
    Pet boots = new Pet { Name = "Boots", Owner = terry };
    Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };
    Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

    List<Person> people = new List<Person> { magnus, terry, charlotte };
    List<Pet> pets = new List<Pet> { barley, boots, whiskers, daisy };

    // Create a list of Person-Pet pairs where
    // each element is an anonymous type that contains a
    // Pet's name and the name of the Person that owns the Pet.
    var query =
        people.Join(pets,
                    person => person,
                    pet => pet.Owner,
                    (person, pet) =>
                        new { OwnerName = person.Name, Pet = pet.Name });

    foreach (var obj in query)
    {
```

```
        Console.WriteLine(
            "{0} - {1}",
            obj.OwnerName,
            obj.Pet);
    }
}

/*
 This code produces the following output:

 Hedlund, Magnus - Daisy
 Adams, Terry - Barley
 Adams, Terry - Boots
 Weiss, Charlotte - Whiskers
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The default equality comparer, Default, is used to hash and compare keys.

A join refers to the operation of correlating the elements of two sources of information based on a common key. Join brings the two information sources and the keys by which they are matched together in one method call. This differs from the use of `SelectMany`, which requires more than one method call to perform the same operation.

Join preserves the order of the elements of `outer`, and for each of these elements, the order of the matching elements of `inner`.

In query expression syntax, a `join` (C#) or `Join` (Visual Basic) clause translates to an invocation of Join.

In relational database terms, the Join method implements an inner equijoin. 'Inner' means that only elements that have a match in the other sequence are included in the results. An 'equijoin' is a join in which the keys are compared for equality. A left outer join operation has no dedicated standard query operator, but can be performed by using the GroupJoin method. For more information, see Join operations.

See also

- Join operations (C#)
- Join operations (Visual Basic)
- join clause (C# Reference)
- Join Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Join<TOuter,TInner,TKey,TResult> (IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)

Source: Join.cs

Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Join<TOuter,TInner,TKey,TResult> (this
System.Collections.Generic.IEnumerable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
```

```
Func<TOuter,TInner,TResult> resultSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IEnumerable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,TInner,TResult>

A function to create a result element from two matching elements.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to hash and compare keys.

Returns

IEnumerable<TResult>

An IEnumerable<T> that has elements of type `TResult` that are obtained by performing an inner join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

If `comparer` is `null`, the default equality comparer, Default, is used to hash and compare keys.

A join refers to the operation of correlating the elements of two sources of information based on a common key. Join brings the two information sources and the keys by which they are matched together in one method call. This differs from the use of `SelectMany`, which requires more than one method call to perform the same operation.

Join preserves the order of the elements of `outer`, and for each of these elements, the order of the matching elements of `inner`.

In relational database terms, the Join method implements an inner equijoin. 'Inner' means that only elements that have a match in the other sequence are included in the results. An 'equijoin' is a join in which the keys are compared for equality. A left outer join operation has no dedicated standard query operator, but can be performed by using the GroupJoin method. For more information, see Join operations.

See also

- Join operations (C#)
- Join operations (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Last Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the last element of a sequence.

Overloads

Expand table

Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.

Last<TSource>(IEnumerable<TSource>)

Source: Last.cs

Returns the last element of a sequence.

C#

```
public static TSource Last<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

An IEnumerable<T> to return the last element of.

Returns

TSource

The value at the last position in the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The source sequence is empty.

Examples

The following code example demonstrates how to use Last<TSource>(IEnumerable<TSource>) to return the last element of an array.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                    83, 23, 87, 67, 12, 19 };

int last = numbers.Last();

Console.WriteLine(last);

/*
 This code produces the following output:

 19
*/
```

Remarks

The Last<TSource>(IEnumerable<TSource>) method throws an exception if `source` contains no elements. To instead return a default value when the source sequence is empty, use the LastOrDefault method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Last.cs

Returns the last element of a sequence that satisfies a specified condition.

C#

```
public static TSource Last<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

The last element in the sequence that passes the test in the specified predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

No element satisfies the condition in `predicate`.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use Last<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to return the last element of an array that satisfies a condition.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                    83, 23, 87, 67, 12, 19 };

int last = numbers.Last(num => num > 80);

Console.WriteLine(last);

/*
 This code produces the following output:

 87
*/
```

Remarks

The Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>) method throws an exception if no matching element is found in `source`. To instead return a default value when no matching element is found, use the LastOrDefault method.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.LastOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the last element of a sequence, or a default value if no element is found.

Overloads

Expand table

LastOrDefault<TSource> (IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource> (IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.

LastOrDefault<TSource> (IEnumerable<TSource>)

Source: Last.cs

Returns the last element of a sequence, or a default value if the sequence contains no elements.

C#

```
public static TSource? LastOrDefault<TSource> (this
```

```
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return the last element of.

Returns

TSource

`default(TSource)` if the source sequence is empty; otherwise, the last element in the IEnumerable<T>.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use LastOrDefault<TSource> (IEnumerable<TSource>) on an empty array.

C#

```
string[] fruits = { };
string last = fruits.LastOrDefault();
Console.WriteLine(
    String.IsNullOrEmpty(last) ? "<string is null or empty>" : last);

/*
 This code produces the following output:

 <string is null or empty>
*/
```

Sometimes the value of `default(TSource)` is not the default value that you want to use if the collection contains no elements. Instead of checking the result for the

unwanted default value and then changing it if necessary, you can use the DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) method to specify the default value that you want to use if the collection is empty. Then, call Last<TSource>(IEnumerable<TSource>) to obtain the last element. The following code example uses both techniques to obtain a default value of 1 if a collection of numeric days of the month is empty. Because the default value for an integer is 0, which does not correspond to any day of the month, the default value must be specified as 1 instead. The first result variable is checked for the unwanted default value after the query has finished executing. The second result variable is obtained by using DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) to specify a default value of 1.

C#

```
List<int> daysOfMonth = new List<int> { };

// Setting the default value to 1 after the query.
int lastDay1 = daysOfMonth.LastOrDefault();
if (lastDay1 == 0)
{
    lastDay1 = 1;
}
Console.WriteLine("The value of the lastDay1 variable is {0}", lastDay1);

// Setting the default value to 1 by using DefaultIfEmpty() in the query.
int lastDay2 = daysOfMonth.DefaultIfEmpty(1).Last();
Console.WriteLine("The value of the lastDay2 variable is {0}", lastDay2);

/*
 This code produces the following output:

 The value of the lastDay1 variable is 1
 The value of the lastDay2 variable is 1
*/
```

Remarks

The default value for reference and nullable types is `null`.

The LastOrDefault method does not provide a way to specify a default value. If you want to specify a default value other than `default(TSource)`, use the DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) method as described in the Example section.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

LastOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Last.cs

Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

C#

```
public static TSource? LastOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

`default(TSource)` if the sequence is empty or if no elements pass the test in the predicate function; otherwise, the last element that passes the test in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use LastOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) by passing in a predicate. In the second call to the method, there is no element in the sequence that satisfies the condition.

C#

```
double[] numbers = { 49.6, 52.3, 51.0, 49.4, 50.2, 48.3 };

double last50 = numbers.LastOrDefault(n => Math.Round(n) == 50.0);

Console.WriteLine("The last number that rounds to 50 is {0}.", last50);

double last40 = numbers.LastOrDefault(n => Math.Round(n) == 40.0);

Console.WriteLine(
    "The last number that rounds to 40 is {0}.",
    last40 == 0.0 ? "<DOES NOT EXIST>" : last40.ToString());

/*
 This code produces the following output:

 The last number that rounds to 50 is 50.2.
 The last number that rounds to 40 is <DOES NOT EXIST>.
*/
```

Remarks

The default value for reference and nullable types is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

LastOrDefault<TSource> (IEnumerable<TSource>, TSource)

Source: Last.cs

Returns the last element of a sequence, or a specified default value if the sequence contains no elements.

C#

```
public static TSource LastOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return the last element of.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if the source sequence is empty; otherwise, the last element in the IEnumerable<T>.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

LastOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>, TSource)

Source: Last.cs

Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.

C#

```
public static TSource LastOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if the sequence is empty or if no elements pass the test in the predicate function; otherwise, the last element that passes the test in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.LongCount Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns an Int64 that represents the number of elements in a sequence.

Overloads

Expand table

LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.

LongCount<TSource>(IEnumerable<TSource>)

Source: Count.cs

Returns an Int64 that represents the total number of elements in a sequence.

C#

```
public static long LongCount<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

An IEnumerable<T> that contains the elements to be counted.

Returns

Int64

The number of elements in the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The number of elements exceeds Int64.MaxValue.

Examples

The following code example demonstrates how to use LongCount<TSource> (IEnumerable<TSource>) to count the elements in an array.

C#

```
string[] fruits = { "apple", "banana", "mango",
                    "orange", "passionfruit", "grape" };

long count = fruits.LongCount();

Console.WriteLine("There are {0} fruits in the collection.", count);

/*
 This code produces the following output:

 There are 6 fruits in the collection.
*/
```

Remarks

Use this method rather than Count when you expect the result to be greater than MaxValue.

In Visual Basic query expression syntax, an `Aggregate Into LongCount()` clause translates to an invocation of LongCount.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Count.cs

Returns an Int64 that represents how many elements in a sequence satisfy a condition.

C#

```
public static long LongCount<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> that contains the elements to be counted.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Int64

A number that represents how many elements in the sequence satisfy the condition in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

OverflowException

The number of matching elements exceeds Int64.MaxValue.

Examples

The following code example demonstrates how to use LongCount<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to count the elements in an array that satisfy a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void LongCountEx2()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    const int Age = 3;

    long count = pets.LongCount(pet => pet.Age > Age);

    Console.WriteLine("There are {0} animals over age {1}.", count, Age);
}

/*
 This code produces the following output:

 There are 2 animals over age 3.
*/
```

Remarks

Use this method rather than Count when you expect the result to be greater than MaxValue.

In Visual Basic query expression syntax, an `Aggregate Into LongCount()` clause translates to an invocation of LongCount.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Max Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the maximum value in a sequence of values.

Overloads

Expand table

Max(IEnumerable<Nullable<Int32>>)	Returns the maximum value in a sequence of nullable Int32 values.
Max(IEnumerable<Single>)	Returns the maximum value in a sequence of Single values.
Max(IEnumerable<Nullable<Single>>)	Returns the maximum value in a sequence of nullable Single values.
Max(IEnumerable<Nullable<Int64>>)	Returns the maximum value in a sequence of nullable Int64 values.
Max(IEnumerable<Nullable<Double>>)	Returns the maximum value in a sequence of nullable Double values.
Max(IEnumerable<Double>)	Returns the maximum value in a sequence of Double values.
Max(IEnumerable<Int64>)	Returns the maximum value in a sequence of Int64 values.
Max(IEnumerable<Int32>)	Returns the maximum value in a sequence of Int32 values.
Max(IEnumerable<Decimal>)	Returns the maximum value in a sequence of Decimal values.
Max(IEnumerable<Nullable<Decimal>>)	Returns the maximum value in a sequence of nullable Decimal values.

Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.

Max(IEnumerable<Nullable<Int32>>)

Source: Max.cs

Returns the maximum value in a sequence of nullable Int32 values.

```
C#

public static int? Max (this System.Collections.Generic.IEnumerable<int?>
source);
```

Parameters

source IEnumerable<Nullable<Int32>>

A sequence of nullable Int32 values to determine the maximum value of.

Returns

Nullable<Int32>

A value of type `Nullable<Int32>` in C# or `Nullable(Of Int32)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Max(IEnumerable<Nullable<Int32>>) method uses the Int32 implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Single>)

Source: Max.cs

Returns the maximum value in a sequence of Single values.

C#

```
public static float Max (this
System.Collections.Generic.IEnumerable<float> source);
```

Parameters

`source` IEnumerable<Single>

A sequence of Single values to determine the maximum value of.

Returns

Single

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Max(IEnumerable<Single>) method uses the Single implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Nullable<Single>>)

Source: Max.cs

Returns the maximum value in a sequence of nullable Single values.

C#

```
public static float? Max (this
System.Collections.Generic.IEnumerable<float?> source);
```

Parameters

`source` IEnumerable<Nullable<Single>>

A sequence of nullable Single values to determine the maximum value of.

Returns

Nullable<Single>

A value of type `Nullable<Single>` in C# or `Nullable(Of Single)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Max(IEnumerable<Nullable<Single>>) method uses the Single implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Max(IEnumerable<Nullable<Int64>>)

Source: Max.cs

Returns the maximum value in a sequence of nullable Int64 values.

```
C#

public static long? Max (this
System.Collections.Generic.IEnumerable<long?> source);
```

Parameters

`source` IEnumerable<Nullable<Int64>>

A sequence of nullable Int64 values to determine the maximum value of.

Returns

Nullable<Int64>

A value of type `Nullable<Int64>` in C# or `Nullable(Of Int64)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Max(IEnumerable<Nullable<Int64>>) method uses the Int64 implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Nullable<Double>>)

Source: Max.cs

Returns the maximum value in a sequence of nullable Double values.

C#

```
public static double? Max (this
System.Collections.Generic.IEnumerable<double?> source);
```

Parameters

`source` IEnumerable<Nullable<Double>>

A sequence of nullable Double values to determine the maximum value of.

Returns

Nullable<Double>

A value of type `Nullable<Double>` in C# or `Nullable(Of Double)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Max(IEnumerable<Nullable<Double>>) to determine the maximum value in a sequence.

```
C#

double?[] doubles = { null, 1.5E+104, 9E+103, -2E+103 };

double? max = doubles.Max();

Console.WriteLine("The largest number is {0}.", max);

/*
 This code produces the following output:

 The largest number is 1.5E+104.
*/
```

Remarks

The Max(IEnumerable<Nullable<Double>>) method uses the Double implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Double>)

Source: Max.cs

Returns the maximum value in a sequence of Double values.

```
C#

public static double Max (this
System.Collections.Generic.IEnumerable<double> source);
```

Parameters

`source` IEnumerable<Double>

A sequence of Double values to determine the maximum value of.

Returns

Double

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Max(IEnumerable<Double>) method uses the Double implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Int64>)

Source: Max.cs

Returns the maximum value in a sequence of Int64 values.

C#

```
public static long Max (this System.Collections.Generic.IEnumerable<long>
source);
```

Parameters

source IEnumerable<Int64>

A sequence of Int64 values to determine the maximum value of.

Returns

Int64

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max(IEnumerable<Int64>) to determine the maximum value in a sequence.

C#

```
List<long> longs = new List<long> { 4294967296L, 466855135L, 81125L };

long max = longs.Max();

Console.WriteLine("The largest number is {0}.", max);

/*
 This code produces the following output:

 The largest number is 4294967296.
*/
```

Remarks

The Max(IEnumerable<Int64>) method uses the Int64 implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Int32>)

Source: Max.cs

Returns the maximum value in a sequence of Int32 values.

```
C#

public static int Max (this System.Collections.Generic.IEnumerable<int>
source);
```

Parameters

`source` IEnumerable<Int32>

A sequence of Int32 values to determine the maximum value of.

Returns

Int32

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Max(IEnumerable<Int32>) method uses the Int32 implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max(IEnumerable<Decimal>)

Source: Max.cs

Returns the maximum value in a sequence of Decimal values.

C#

```
public static decimal Max (this
System.Collections.Generic.IEnumerable<decimal> source);
```

Parameters

`source` IEnumerable<Decimal>

A sequence of Decimal values to determine the maximum value of.

Returns

Decimal

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Max(IEnumerable<Decimal>) method uses the Decimal implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Max(IEnumerable<Nullable<Decimal>>)

Source: Max.cs

Returns the maximum value in a sequence of nullable Decimal values.

```
C#

public static decimal? Max (this
System.Collections.Generic.IEnumerable<decimal?> source);
```

Parameters

`source` IEnumerable<Nullable<Decimal>>

A sequence of nullable Decimal values to determine the maximum value of.

Returns

Nullable<Decimal>

A value of type `Nullable<Decimal>` in C# or `Nullable(Of Decimal)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Max(IEnumerable<Nullable<Decimal>>) method uses the Decimal implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource,TResult> (IEnumerable<TSource>, Func<TSource,TResult>)

Source: Max.cs

Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.

C#

```
public static TResult? Max<TSource,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TResult> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the value returned by `selector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,TResult>

A transform function to apply to each element.

Returns

TResult

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
```

```
    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

If type `TResult` implements IComparable<T>, this method uses that implementation to compare values. Otherwise, if type `TResult` implements IComparable, that implementation is used to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- MaxBy
- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum Single value.

```
C#

public static float Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>) method uses the Single implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type,

specifically Single.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.

C#

```
public static float? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The value that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
```

```
    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>) method uses the Single implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Single>` in C# or `Nullable(Of Single)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.

```
C#

public static long? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Int64>

The value that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in

a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>) method uses the Int64 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Int64>` in C# or `Nullable(Of Int64)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.

C#

```
public static int? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int?
> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Int32>

The value of type `Nullable<Int32>` in C# or `Nullable(Of Int32)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
```

```
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>) method uses the Int32 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Int32>` in C# or `Nullable(Of Int32)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.

```
C#

public static decimal? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The value of type `Nullable<Decimal>` in C# or `Nullable(Of Decimal)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>) method uses the Decimal implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Decimal>` in C# or `Nullable(Of Decimal)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.

C#

```
public static double? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The value of type `Nullable<Double>` in C# or `Nullable(Of Double)` in Visual Basic that corresponds to the maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}
```

```
/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>) method uses the Double implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Double>` in C# or `Nullable(Of Double)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum Int32 value.

```
C#

public static int Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Int32

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>) method uses the Int32 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Int32.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum Double value.

C#

```
public static double Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:
```

```
 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>) method uses the Double implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Double.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum Decimal value.

```C#
public static decimal Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

Note

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>) method uses the Decimal implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Decimal.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)

Source: Max.cs

Returns the maximum value in a generic sequence.

C#

```
public static TSource? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
System.Collections.Generic.IComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`comparer` IComparer<TSource>

The IComparer<T> to compare values.

Returns

TSource

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No object in `source` implements the IComparable or IComparable<T> interface.

Remarks

If type `TSource` implements IComparable<T>, the Max<TSource> (IEnumerable<TSource>) method uses that implementation to compare values. Otherwise, if type `TSource` implements IComparable, that implementation is used to compare values.

If `TSource` is a reference type and the source sequence is empty or contains only values that are `null`, this method returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Max<TSource>(IEnumerable<TSource>)

Source: Max.cs

Returns the maximum value in a generic sequence.

```
C#
```

```
public static TSource? Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

Returns

TSource

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No object in `source` implements the IComparable or IComparable<T> interface.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>) to determine the maximum value in a sequence of IComparable<T> objects.

C#

```
/// <summary>
/// This class implements IComparable to be able to
/// compare one Pet to another Pet.
/// </summary>
class Pet : IComparable<Pet>
{
    public string Name { get; set; }
    public int Age { get; set; }
```

```
    /// <summary>
    /// Compares this Pet to another Pet by
    /// summing each Pet's age and name length.
    /// </summary>
    /// <param name="other">The Pet to compare this Pet to.</param>
    /// <returns>-1 if this Pet is 'less' than the other Pet,
    /// 0 if they are equal,
    /// or 1 if this Pet is 'greater' than the other Pet.</returns>
    int IComparable<Pet>.CompareTo(Pet other)
    {
        int sumOther = other.Age + other.Name.Length;
        int sumThis = this.Age + this.Name.Length;

        if (sumOther > sumThis)
            return -1;
        else if (sumOther == sumThis)
            return 0;
        else
            return 1;
    }
}

public static void MaxEx3()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    Pet max = pets.Max();

    Console.WriteLine(
        "The 'maximum' animal is {0}.",
        max.Name);
}

/*
 This code produces the following output:

 The 'maximum' animal is Barley.
*/
```

Remarks

If type `TSource` implements IComparable<T>, the Max<TSource> (IEnumerable<TSource>) method uses that implementation to compare values. Otherwise, if type `TSource` implements IComparable, that implementation is used to compare values.

If `TSource` is a reference type and the source sequence is empty or contains only values that are `null`, this method returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)

Source: Max.cs

Invokes a transform function on each element of a sequence and returns the maximum Int64 value.

C#

```
public static long Max<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Int64

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the maximum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
```

```
}

public static void MaxEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int max = pets.Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
 This code produces the following output:

 The maximum pet age plus name length is 14.
*/
```

Remarks

The Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>) method uses the Int64 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Int64.

In Visual Basic query expression syntax, an `Aggregate Into Max()` clause translates to an invocation of Max.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.MaxBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.

MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)

Source: Max.cs

Returns the maximum value in a generic sequence according to a specified key selector function.

C#

```
public static TSource? MaxBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

TKey

The type of key to compare elements by.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

TSource

The value with the maximum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

InvalidOperationException

`TSource` is a primitive type and the source sequence is empty.

Remarks

If the source sequence is empty, two possible outcomes are possible depending on the source type. If `TSource` is a nullable type, this method returns `null`. If `TSource` is a non-nullable struct, such as a primitive type, an InvalidOperationException is thrown.

If the source sequence contains only values that are `null`, this method returns `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

MaxBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: Max.cs

Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.

```
C#

public static TSource? MaxBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of key to compare elements by.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the maximum value of.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`comparer` IComparer<TKey>

The IComparer<T> to compare keys.

Returns

TSource

The value with the maximum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

InvalidOperationException

`TSource` is a primitive type and the source sequence is empty.

Remarks

If the source sequence is empty, two possible outcomes are possible depending on the source type. If `TSource` is a nullable type, this method returns `null`. If `TSource` is a non-nullable struct, such as a primitive type, an InvalidOperationException is thrown.

If the source sequence contains only values that are `null`, this method returns `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.Min Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns the minimum value in a sequence of values.

Overloads

Expand table

Min(IEnumerable<Nullable<Int32>>)	Returns the minimum value in a sequence of nullable Int32 values.
Min(IEnumerable<Single>)	Returns the minimum value in a sequence of Single values.
Min(IEnumerable<Nullable<Single>>)	Returns the minimum value in a sequence of nullable Single values.
Min(IEnumerable<Nullable<Int64>>)	Returns the minimum value in a sequence of nullable Int64 values.
Min(IEnumerable<Nullable<Double>>)	Returns the minimum value in a sequence of nullable Double values.
Min(IEnumerable<Decimal>)	Returns the minimum value in a sequence of Decimal values.
Min(IEnumerable<Int64>)	Returns the minimum value in a sequence of Int64 values.
Min(IEnumerable<Int32>)	Returns the minimum value in a sequence of Int32 values.
Min(IEnumerable<Double>)	Returns the minimum value in a sequence of Double values.
Min(IEnumerable<Nullable<Decimal>>)	Returns the minimum value in a sequence of nullable Decimal values.

Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.

Min(IEnumerable<Nullable<Int32>>)

Source: Min.cs

Returns the minimum value in a sequence of nullable Int32 values.

```
C#

public static int? Min (this System.Collections.Generic.IEnumerable<int?>
source);
```

Parameters

source IEnumerable<Nullable<Int32>>

A sequence of nullable Int32 values to determine the minimum value of.

Returns

Nullable<Int32>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Min(IEnumerable<Nullable<Int32>>) to determine the minimum value in a sequence.

```
C#

int?[] grades = { 78, 92, null, 99, 37, 81 };

int? min = grades.Min();

Console.WriteLine("The lowest grade is {0}.", min);

/*
 This code produces the following output:
```

```
 The lowest grade is 37.
*/
```

Remarks

The Min(IEnumerable<Nullable<Int32>>) method uses the Int32 implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Single>)

Source: Min.cs

Returns the minimum value in a sequence of Single values.

C#

```
public static float Min (this
```

```
System.Collections.Generic.IEnumerable<float> source);
```

Parameters

`source` IEnumerable<Single>

A sequence of Single values to determine the minimum value of.

Returns

Single

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Min(IEnumerable<Single>) method uses the Single implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Nullable<Single>>)

Source: Min.cs

Returns the minimum value in a sequence of nullable Single values.

```
C#

public static float? Min (this
System.Collections.Generic.IEnumerable<float?> source);
```

Parameters

`source` IEnumerable<Nullable<Single>>

A sequence of nullable Single values to determine the minimum value of.

Returns

Nullable<Single>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Min(IEnumerable<Nullable<Single>>) method uses the Single implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Nullable<Int64>>)

Source: Min.cs

Returns the minimum value in a sequence of nullable Int64 values.

C#

```
public static long? Min (this
System.Collections.Generic.IEnumerable<long?> source);
```

Parameters

`source` IEnumerable<Nullable<Int64>>

A sequence of nullable Int64 values to determine the minimum value of.

Returns

Nullable<Int64>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Min(IEnumerable<Nullable<Int64>>) method uses the Int64 implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Nullable<Double>>)

Source: Min.cs

Returns the minimum value in a sequence of nullable Double values.

```
C#
```

```
public static double? Min (this
System.Collections.Generic.IEnumerable<double?> source);
```

Parameters

`source` IEnumerable<Nullable<Double>>

A sequence of nullable Double values to determine the minimum value of.

Returns

Nullable<Double>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Min(IEnumerable<Nullable<Double>>) method uses the Double implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Decimal>)

Source: Min.cs

Returns the minimum value in a sequence of Decimal values.

C#

```
public static decimal Min (this
System.Collections.Generic.IEnumerable<decimal> source);
```

Parameters

`source` IEnumerable<Decimal>

A sequence of Decimal values to determine the minimum value of.

Returns

Decimal

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Min(IEnumerable<Decimal>) method uses the Decimal implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Int64>)

Source: Min.cs

Returns the minimum value in a sequence of Int64 values.

C#

```
public static long Min (this System.Collections.Generic.IEnumerable<long>
source);
```

Parameters

`source` IEnumerable<Int64>

A sequence of Int64 values to determine the minimum value of.

Returns

Int64

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Min(IEnumerable<Int64>) method uses the Int64 implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Int32>)

Source: Min.cs

Returns the minimum value in a sequence of Int32 values.

```
C#
```

```
public static int Min (this System.Collections.Generic.IEnumerable<int>
source);
```

Parameters

`source` IEnumerable<Int32>

A sequence of Int32 values to determine the minimum value of.

Returns

Int32

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Remarks

The Min(IEnumerable<Int32>) method uses the Int32 implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Double>)

Source: Min.cs

Returns the minimum value in a sequence of Double values.

C#

```
public static double Min (this
System.Collections.Generic.IEnumerable<double> source);
```

Parameters

`source` IEnumerable<Double>

A sequence of Double values to determine the minimum value of.

Returns

Double

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min(IEnumerable<Double>) to determine the minimum value in a sequence.

C#

```
double[] doubles = { 1.5E+104, 9E+103, -2E+103 };

double min = doubles.Min();

Console.WriteLine("The smallest number is {0}.", min);

/*
 This code produces the following output:

 The smallest number is -2E+103.
*/
```

Remarks

The Min(IEnumerable<Double>) method uses the Double implementation of IComparable<T> to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min(IEnumerable<Nullable<Decimal>>)

Source: Min.cs

Returns the minimum value in a sequence of nullable Decimal values.

```
C#

public static decimal? Min (this
System.Collections.Generic.IEnumerable<decimal?> source);
```

Parameters

source IEnumerable<Nullable<Decimal>>

A sequence of nullable Decimal values to determine the minimum value of.

Returns

Nullable<Decimal>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Min(IEnumerable<Nullable<Decimal>>) method uses the Decimal implementation of IComparable<T> to compare values.

If the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource,TResult> (IEnumerable<TSource>, Func<TSource,TResult>)

Source: Min.cs

Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.

C#

```
public static TResult? Min<TSource,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TResult> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the value returned by `selector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,TResult>

A transform function to apply to each element.

Returns

TResult

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
```

```
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

If type `TResult` implements IComparable<T>, this method uses that implementation to compare values. Otherwise, if type `TResult` implements IComparable, that implementation is used to compare values.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- MinBy
- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum Single value.

```
C#

public static float Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>) method uses the Single implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Single.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.

C#

```
public static float? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:
```

```
 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>) method uses the Single implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Single>` in C# or `Nullable(Of Single)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.

```C#
public static long? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Int64>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>) method uses the Int64 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Int64>` in C# or `Nullable(Of Int64)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.

C#

```
public static double? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>) method uses the Double implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Double>` in C# or `Nullable(Of Double)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.

C#

```
public static decimal? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>) method uses the Decimal implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Decimal>` in C# or `Nullable(Of Decimal)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.

```
C#

public static int? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int?
> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Int32>

The value that corresponds to the minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>) method uses the Int32 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of source into a numeric type, specifically `Nullable<Int32>` in C# or `Nullable(Of Int32)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum Int32 value.

C#

```
public static int Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Int32

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
```

```
    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>) method uses the Int32 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Int32.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum Double value.

```
C#

public static double Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>) method uses the Double implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Double.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum Decimal value.

C#

```
public static decimal Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
```

```
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>) method uses the Decimal implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Decimal.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)

Source: Min.cs

Returns the minimum value in a generic sequence.

```
C#

public static TSource? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
System.Collections.Generic.IComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`comparer` IComparer<TSource>

The IComparer<T> to compare values.

Returns

TSource

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No object in `source` implements the IComparable or IComparable<T> interface.

Remarks

If type `TSource` implements IComparable<T>, the Max<TSource> (IEnumerable<TSource>) method uses that implementation to compare values. Otherwise, if type `TSource` implements IComparable, that implementation is used to compare values.

If `TSource` is a reference type and the source sequence is empty or contains only values that are `null`, this method returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Min<TSource>(IEnumerable<TSource>)

Source: Min.cs

Returns the minimum value in a generic sequence.

```
C#

public static TSource? Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

Returns

TSource

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No object in `source` implements the IComparable or IComparable<T> interface.

Examples

The following code example demonstrates how to use Min<TSource> (IEnumerable<TSource>) to determine the minimum value in a sequence of IComparable<T> objects.

C#

```
/// <summary>
/// This class implements IComparable in order to
/// be able to compare different Pet objects.
/// </summary>
class Pet : IComparable<Pet>
{
    public string Name { get; set; }
    public int Age { get; set; }

    /// <summary>
    /// Compares this Pet's age to another Pet's age.
    /// </summary>
    /// <param name="other">The Pet to compare this Pet to.</param>
    /// <returns>-1 if this Pet's age is smaller,
    /// 0 if the Pets' ages are equal, or
    /// 1 if this Pet's age is greater.</returns>
    int IComparable<Pet>.CompareTo(Pet other)
    {
        if (other.Age > this.Age)
            return -1;
        else if (other.Age == this.Age)
            return 0;
        else
            return 1;
    }
}

public static void MinEx3()
{
```

```
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    Pet min = pets.Min();

    Console.WriteLine(
        "The 'minimum' animal is {0}.",
        min.Name);
}

/*
 This code produces the following output:

 The 'minimum' animal is Whiskers.
*/
```

Remarks

If type `TSource` implements IComparable<T>, this method uses that implementation to compare values. Otherwise, if type `TSource` implements IComparable, that implementation is used to compare values.

If `TSource` is a reference type and the source sequence is empty or contains only values that are `null`, this function returns `null`.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)

Source: Min.cs

Invokes a transform function on each element of a sequence and returns the minimum Int64 value.

```
C#

public static long Min<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Int64

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>) to determine the minimum value in a sequence of projected values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx4()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    int min = pets.Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
 This code produces the following output:

 The youngest animal is age 1.
*/
```

Remarks

The Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>) method uses the Int64 implementation of IComparable<T> to compare values.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Int64.

In Visual Basic query expression syntax, an `Aggregate Into Min()` clause translates to an invocation of Min.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.MinBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.

MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)

Source: Min.cs

Returns the minimum value in a generic sequence according to a specified key selector function.

C#

```
public static TSource? MinBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

TKey

The type of key to compare elements by.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

TSource

The value with the minimum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

InvalidOperationException

`TSource` is a primitive type and the source sequence is empty.

Remarks

If the source sequence is empty, two possible outcomes are possible depending on the source type. If `TSource` is a nullable type, this method returns `null`. If `TSource` is a non-nullable struct, such as a primitive type, an InvalidOperationException is thrown.

If the source sequence contains only values that are `null`, this method returns `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: Min.cs

Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.

C#

```
public static TSource? MinBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of key to compare elements by.

Parameters

`source` IEnumerable<TSource>

A sequence of values to determine the minimum value of.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`comparer` IComparer<TKey>

The IComparer<T> to compare keys.

Returns

TSource

The value with the minimum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

InvalidOperationException

`TSource` is a primitive type and the source sequence is empty.

Remarks

If the source sequence is empty, two possible outcomes are possible depending on the source type. If `TSource` is a nullable type, this method returns `null`. If `TSource` is a non-nullable struct, such as a primitive type, an InvalidOperationException is thrown.

If the source sequence contains only values that are `null`, this method returns `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.OfType<TResult> (IEnumerable) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Cast.cs

Filters the elements of an IEnumerable based on a specified type.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
OfType<TResult> (this System.Collections.IEnumerable source);
```

Type Parameters

`TResult`

The type to filter the elements of the sequence on.

Parameters

`source` IEnumerable

The IEnumerable whose elements to filter.

Returns

IEnumerable<TResult>

An IEnumerable<T> that contains elements from the input sequence of type `TResult`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use OfType to filter the elements of an IEnumerable.

C#

```
System.Collections.ArrayList fruits = new()
{
    "Mango",
    "Orange",
    null,
    "Apple",
    3.0,
    "Banana"
};

// Apply OfType() to the ArrayList.
IEnumerable<string> query1 = fruits.OfType<string>();

Console.WriteLine("Elements of type 'string' are:");
foreach (string fruit in query1)
{
    Console.WriteLine(fruit);
}

// The following query shows that the standard query operators such as
// Where() can be applied to the ArrayList type after calling OfType().
IEnumerable<string> query2 =
    fruits.OfType<string>().Where(fruit =>
    fruit.Contains('n', StringComparison.CurrentCultureIgnoreCase));

Console.WriteLine("\nThe following strings contain 'n':");
foreach (string fruit in query2)
{
    Console.WriteLine(fruit);
}

// This code produces the following output:
//
// Elements of type 'string' are:
// Mango
// Orange
// Apple
// Banana
//
// The following strings contain 'n':
// Mango
// Orange
// Banana
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The OfType<TResult>(IEnumerable) method returns only those elements in `source` that are non-null and compatible with type `TResult`. To receive an exception if an element cannot be cast to type `TResult`, use Cast<TResult>(IEnumerable).

This method is one of the few standard query operator methods that can be applied to a collection that has a non-parameterized type, such as an ArrayList. This is because OfType extends the type IEnumerable. OfType cannot only be applied to collections that are based on the parameterized IEnumerable<T> type, but collections that are based on the non-parameterized IEnumerable type also.

By applying OfType to a collection that implements IEnumerable, you gain the ability to query the collection by using the standard query operators. For example, specifying a type argument of Object to OfType would return an object of type `IEnumerable<Object>` in C# or `IEnumerable(Of Object)` in Visual Basic, to which the standard query operators can be applied.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Order Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.

Order<T>(IEnumerable<T>)

Source: OrderBy.cs

Sorts the elements of a sequence in ascending order.

C#

```
public static System.Linq.IOrderedEnumerable<T> Order<T> (this
System.Collections.Generic.IEnumerable<T> source);
```

Type Parameters

T

The type of the elements of `source`.

Parameters

source IEnumerable<T>

A sequence of values to order.

Returns

IOrderedEnumerable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated by calling its <see cref="M:System.Collections.Generic.IEnumerable`1.GetEnumerator> method.

This method compares elements by using the default comparer <see cref="P:System.Collections.Generic.Comparer`1.Default>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Order<T>(IEnumerable<T>, IComparer<T>)

Source: OrderBy.cs

Sorts the elements of a sequence in ascending order.

C#

```
public static System.Linq.IOrderedEnumerable<T> Order<T> (this
System.Collections.Generic.IEnumerable<T> source,
System.Collections.Generic.IComparer<T>? comparer);
```

Type Parameters

`T`

The type of the elements of `source`.

Parameters

`source` IEnumerable<T>

A sequence of values to order.

`comparer` IComparer<T>

An IComparer<T> to compare keys.

Returns

IOrderedEnumerable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated by calling its <see cref="M:System.Collections.Generic.IEnumerable`1.GetEnumerator> method.

If comparer is `null`, the default comparer <see cref="P:System.Collections.Generic.Comparer`1.Default> is used to compare elements.

See also

- OrderBy

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Enumerable.OrderBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Sorts the elements of a sequence in ascending order.

Overloads

Expand table

OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.

OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)

Source: OrderBy.cs

Sorts the elements of a sequence in ascending order according to a key.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
OrderBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use OrderBy<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>) to sort the elements of a sequence.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void OrderByEx1()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    IEnumerable<Pet> query = pets.OrderBy(pet => pet.Age);

    foreach (Pet pet in query)
    {
        Console.WriteLine("{0} - {1}", pet.Name, pet.Age);
```

```
    }
}

/*
 This code produces the following output:

 Whiskers - 1
 Boots - 4
 Barley - 8
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

Two methods are defined to extend the type IOrderedEnumerable<TElement>, which is the return type of this method. These two methods, namely `ThenBy` and `ThenByDescending`, enable you to specify additional sort criteria to sort a sequence. `ThenBy` and `ThenByDescending` also return an IOrderedEnumerable<TElement>, which means any number of consecutive calls to `ThenBy` or `ThenByDescending` can be made.

> **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

This method compares keys by using the default comparer Default.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

In query expression syntax, an `orderby` (C#) or `Order By` (Visual Basic) clause translates to an invocation of OrderBy.

See also

- OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)
- OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)
- orderby clause (C# Reference)
- Order By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: OrderBy.cs

Sorts the elements of a sequence in ascending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
OrderBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

Two methods are defined to extend the type IOrderedEnumerable<TElement>, which is the return type of this method. These two methods, namely `ThenBy` and

`ThenByDescending`, enable you to specify additional sort criteria to sort a sequence. `ThenBy` and `ThenByDescending` also return an IOrderedEnumerable<TElement>, which means any number of consecutive calls to `ThenBy` or `ThenByDescending` can be made.

> ⓘ **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

If `comparer` is `null`, the default comparer Default is used to compare keys.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

See also

- OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)
- OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.OrderByDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Sorts the elements of a sequence in descending order.

Overloads

Expand table

OrderByDescending<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.

OrderByDescending<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)

Source: OrderBy.cs

Sorts the elements of a sequence in descending order according to a key.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
OrderByDescending<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

For an example of this method, see OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>).

Two methods are defined to extend the type IOrderedEnumerable<TElement>, which is the return type of this method. These two methods, namely `ThenBy` and

`ThenByDescending`, enable you to specify additional sort criteria to sort a sequence. `ThenBy` and `ThenByDescending` also return an IOrderedEnumerable<TElement>, which means any number of consecutive calls to `ThenBy` or `ThenByDescending` can be made.

> ⓘ **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

This method compares keys by using the default comparer Default.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

In query expression syntax, an `orderby descending` (C#) or `Order By Descending` (Visual Basic) clause translates to an invocation of OrderByDescending.

See also

- OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)
- OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)
- orderby clause (C# Reference)
- Order By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

OrderByDescending<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: OrderBy.cs

Sorts the elements of a sequence in descending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
OrderByDescending<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

TSource

The type of the elements of `source`.

TKey

The type of the key returned by `keySelector`.

Parameters

source IEnumerable<TSource>

A sequence of values to order.

keySelector Func<TSource,TKey>

A function to extract a key from an element.

comparer IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>) to sort the elements of a sequence in descending order by using a transform function and a custom comparer.

C#

```
/// <summary>
/// This IComparer class sorts by the fractional part of the decimal
number.
/// </summary>
public class SpecialComparer : IComparer<decimal>
{
    /// <summary>
    /// Compare two decimal numbers by their fractional parts.
    /// </summary>
    /// <param name="d1">The first decimal to compare.</param>
    /// <param name="d2">The second decimal to compare.</param>
    /// <returns>1 if the first decimal's fractional part
    /// is greater than the second decimal's fractional part,
    /// -1 if the first decimal's fractional
    /// part is less than the second decimal's fractional part,
    /// or the result of calling Decimal.Compare()
    /// if the fractional parts are equal.</returns>
    public int Compare(decimal d1, decimal d2)
    {
        decimal fractional1, fractional2;

        // Get the fractional part of the first number.
        try
        {
            fractional1 = decimal.Remainder(d1, decimal.Floor(d1));
        }
        catch (DivideByZeroException)
        {
            fractional1 = d1;
        }
        // Get the fractional part of the second number.
        try
        {
            fractional2 = decimal.Remainder(d2, decimal.Floor(d2));
        }
        catch (DivideByZeroException)
        {
```

```
            fractional2 = d2;
        }

        if (fractional1 == fractional2)
            return Decimal.Compare(d1, d2);
        else if (fractional1 > fractional2)
            return 1;
        else
            return -1;
    }
}

public static void OrderByDescendingEx1()
{
    List<decimal> decimals =
        new List<decimal> { 6.2m, 8.3m, 0.5m, 1.3m, 6.3m, 9.7m };

    IEnumerable<decimal> query =
        decimals.OrderByDescending(num =>
                                   num, new SpecialComparer());

    foreach (decimal num in query)
    {
        Console.WriteLine(num);
    }
}

/*
 This code produces the following output:

 9.7
 0.5
 8.3
 6.3
 1.3
 6.2
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

Two methods are defined to extend the type IOrderedEnumerable<TElement>, which is the return type of this method. These two methods, namely `ThenBy` and `ThenByDescending`, enable you to specify additional sort criteria to sort a sequence. `ThenBy` and `ThenByDescending` also return an IOrderedEnumerable<TElement>, which means any number of consecutive calls to `ThenBy` or `ThenByDescending` can be made.

> **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

If `comparer` is `null`, the default comparer Default is used to compare keys.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

See also

- OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)
- OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.OrderDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.

OrderDescending<T>(IEnumerable<T>, IComparer<T>)

Source: OrderBy.cs

Sorts the elements of a sequence in descending order.

C#

```
public static System.Linq.IOrderedEnumerable<T> OrderDescending<T> (this
System.Collections.Generic.IEnumerable<T> source,
System.Collections.Generic.IComparer<T>? comparer);
```

Type Parameters

T

The type of the elements of `source`.

Parameters

source IEnumerable<T>

A sequence of values to order.

`comparer` IComparer<T>

An IComparer<T> to compare keys.

Returns

IOrderedEnumerable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated by calling its <see cref="M:System.Collections.Generic.IEnumerable`1.GetEnumerator> method.

If comparer is `null`, the default comparer <see cref="P:System.Collections.Generic.Comparer`1.Default> is used to compare elements.

See also

- OrderByDescending

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

OrderDescending<T>(IEnumerable<T>)

Source: OrderBy.cs

Sorts the elements of a sequence in descending order.

```
C#

public static System.Linq.IOrderedEnumerable<T> OrderDescending<T> (this
System.Collections.Generic.IEnumerable<T> source);
```

Type Parameters

T

The type of the elements of `source`.

Parameters

source IEnumerable<T>

A sequence of values to order.

Returns

IOrderedEnumerable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated by calling its <see cref="M:System.Collections.Generic.IEnumerable`1.GetEnumerator> method.

This method compares elements by using the default comparer <see cref="P:System.Collections.Generic.Comparer`1.Default>.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Enumerable.Prepend<TSource> (IEnumerable<TSource>, TSource) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: AppendPrepend.cs

Adds a value to the beginning of the sequence.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Prepend<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, TSource element);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

A sequence of values.

element TSource

The value to prepend to `source`.

Returns

IEnumerable<TSource>

A new sequence that begins with `element`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Prepend to prepend a value to the beginning of the sequence.

C#

```
// Creating a list of numbers
List<int> numbers = new List<int> { 1, 2, 3, 4 };

// Trying to prepend any value of the same type
numbers.Prepend(0);

// It doesn't work because the original list has not been changed
Console.WriteLine(string.Join(", ", numbers));

// It works now because we are using a changed copy of the original list
Console.WriteLine(string.Join(", ", numbers.Prepend(0)));

// If you prefer, you can create a new list explicitly
List<int> newNumbers = numbers.Prepend(0).ToList();

// And then write to the console output
Console.WriteLine(string.Join(", ", newNumbers));

// This code produces the following output:
//
// 1, 2, 3, 4
// 0, 1, 2, 3, 4
// 0, 1, 2, 3, 4
```

Remarks

> **Note**
>
> This method does not modify the elements of the collection. Instead, it creates a copy of the collection with the new element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.6, 2.0, 2.1

Enumerable.Range(Int32, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Range.cs

Generates a sequence of integral numbers within a specified range.

C#

```
public static System.Collections.Generic.IEnumerable<int> Range (int start,
int count);
```

Parameters

`start` Int32

The value of the first integer in the sequence.

`count` Int32

The number of sequential integers to generate.

Returns

IEnumerable<Int32>

An `IEnumerable<Int32>` in C# or `IEnumerable(Of Int32)` in Visual Basic that contains a range of sequential integral numbers.

Exceptions

ArgumentOutOfRangeException

`count` is less than 0.

-or-

`start` + `count` -1 is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Range to generate a sequence of values.

C#

```
// Generate a sequence of integers from 1 to 10
// and then select their squares.
IEnumerable<int> squares = Enumerable.Range(1, 10).Select(x => x * x);

foreach (int num in squares)
{
    Console.WriteLine(num);
}

/*
 This code produces the following output:

 1
 4
 9
 16
 25
 36
 49
 64
 81
 100
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Repeat<TResult>(TResult, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Repeat.cs

Generates a sequence that contains one repeated value.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Repeat<TResult> (TResult element, int count);
```

Type Parameters

TResult

The type of the value to be repeated in the result sequence.

Parameters

element TResult

The value to be repeated.

count Int32

The number of times to repeat the value in the generated sequence.

Returns

IEnumerable<TResult>

An IEnumerable<T> that contains a repeated value.

Exceptions

ArgumentOutOfRangeException

`count` is less than 0.

Examples

The following code example demonstrates how to use Repeat to generate a sequence of a repeated value.

C#

```
IEnumerable<string> strings =
    Enumerable.Repeat("I like programming.", 15);

foreach (String str in strings)
{
    Console.WriteLine(str);
}

/*
 This code produces the following output:

 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Reverse<TSource> (IEnumerable<TSource>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Reverse.cs

Inverts the order of the elements in a sequence.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Reverse<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

A sequence of values to reverse.

Returns

IEnumerable<TSource>

A sequence whose elements correspond to those of the input sequence in reverse order.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Reverse to reverse the order of elements in an array.

C#

```
char[] apple = { 'a', 'p', 'p', 'l', 'e' };

char[] reversed = apple.Reverse().ToArray();

foreach (char chr in reversed)
{
    Console.Write(chr + " ");
}
Console.WriteLine();

/*
 This code produces the following output:

 e l p p a
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

Unlike OrderBy, this sorting method does not consider the actual values themselves in determining the order. Rather, it just returns the elements in the reverse order from which they are produced by the underlying source.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Enumerable.Select Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Projects each element of a sequence into a new form.

Overloads

Expand table

Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.

Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)

Source: Select.cs

Projects each element of a sequence into a new form by incorporating the element's index.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Select<TSource,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,int,TResult> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the value returned by `selector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to invoke a transform function on.

`selector` Func<TSource,Int32,TResult>

A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the transform function on each element of `source`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>) to project over a sequence of values and use the index of each element.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange",
                    "passionfruit", "grape" };

var query =
    fruits.Select((fruit, index) =>
                    new { index, str = fruit.Substring(0, index) });

foreach (var obj in query)
{
```

```
    Console.WriteLine("{0}", obj);
}

/*
 This code produces the following output:

 { index = 0, str =  }
 { index = 1, str = b }
 { index = 2, str = ma }
 { index = 3, str = ora }
 { index = 4, str = pass }
 { index = 5, str = grape }
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The first argument to `selector` represents the element to process. The second argument to `selector` represents the zero-based index of that element in the source sequence. This can be useful if the elements are in a known order and you want to do something with an element at a particular index, for example. It can also be useful if you want to retrieve the index of one or more elements.

This projection method requires the transform function, `selector`, to produce one value for each value in the source sequence, `source`. If `selector` returns a value that is itself a collection, it is up to the consumer to traverse the subsequences manually. In such a situation, it might be better for your query to return a single coalesced sequence of values. To achieve this, use the SelectMany method instead of Select. Although `SelectMany` works similarly to `Select`, it differs in that the transform function returns a collection that is then expanded by `SelectMany` before it is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Select<TSource,TResult> (IEnumerable<TSource>, Func<TSource,TResult>)

Source: Select.cs

Projects each element of a sequence into a new form.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Select<TSource,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TResult> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the value returned by `selector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to invoke a transform function on.

`selector` Func<TSource,TResult>

A transform function to apply to each element.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the transform function on each element of `source`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>) to project over a sequence of values.

C#

```
IEnumerable<int> squares =
    Enumerable.Range(1, 10).Select(x => x * x);

foreach (int num in squares)
{
    Console.WriteLine(num);
}
/*
 This code produces the following output:

 1
 4
 9
 16
 25
 36
 49
 64
 81
 100
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is

enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

This projection method requires the transform function, `selector`, to produce one value for each value in the source sequence, `source`. If `selector` returns a value that is itself a collection, it is up to the consumer to traverse the subsequences manually. In such a situation, it might be better for your query to return a single coalesced sequence of values. To achieve this, use the SelectMany method instead of Select. Although `SelectMany` works similarly to `Select`, it differs in that the transform function returns a collection that is then expanded by `SelectMany` before it is returned.

In query expression syntax, a `select` (C#) or `Select` (Visual Basic) clause translates to an invocation of Select.

See also

- select clause (C# Reference)
- Select Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.SelectMany Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.

Overloads

Expand table

SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)

Source: SelectMany.cs

Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,System.Collections.Generic.IEnumerable<TCollection>>
collectionSelector, Func<TSource,TCollection,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TCollection`

The type of the intermediate elements collected by `collectionSelector`.

`TResult`

The type of the elements of the resulting sequence.

Parameters

`source` IEnumerable<TSource>

A sequence of values to project.

`collectionSelector` Func<TSource,IEnumerable<TCollection>>

A transform function to apply to each element of the input sequence.

`resultSelector` Func<TSource,TCollection,TResult>

A transform function to apply to each element of the intermediate sequence.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the one-to-many transform function `collectionSelector` on each element of `source` and then mapping each of those sequence elements and their corresponding source element to a result element.

Exceptions

ArgumentNullException

`source` or `collectionSelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>) to perform a one-to-many projection over an array and use a result selector function to keep each corresponding element from the source sequence in scope for the final call to `Select`.

C#

```
class PetOwner
{
    public string Name { get; set; }
    public List<string> Pets { get; set; }
}

public static void SelectManyEx3()
{
    PetOwner[] petOwners =
        { new PetOwner { Name="Higa",
              Pets = new List<string>{ "Scruffy", "Sam" } },
          new PetOwner { Name="Ashkenazi",
              Pets = new List<string>{ "Walker", "Sugar" } },
          new PetOwner { Name="Price",
              Pets = new List<string>{ "Scratches", "Diesel" } },
          new PetOwner { Name="Hines",
              Pets = new List<string>{ "Dusty" } } };

    // Project the pet owner's name and the pet's name.
    var query =
        petOwners
        .SelectMany(petOwner => petOwner.Pets, (petOwner, petName) => new
{ petOwner, petName })
```

```
        .Where(ownerAndPet => ownerAndPet.petName.StartsWith("S"))
        .Select(ownerAndPet =>
                new
                {
                    Owner = ownerAndPet.petOwner.Name,
                    Pet = ownerAndPet.petName
                }
        );

    // Print the results.
    foreach (var obj in query)
    {
        Console.WriteLine(obj);
    }
}

// This code produces the following output:
//
// {Owner=Higa, Pet=Scruffy}
// {Owner=Higa, Pet=Sam}
// {Owner=Ashkenazi, Pet=Sugar}
// {Owner=Price, Pet=Scratches}
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>) method is useful when you have to keep the elements of `source` in scope for query logic that occurs after the call to SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>). See the Example section for a code example. If there is a bidirectional relationship between objects of type `TSource` and objects of type `TCollection`, that is, if an object of type `TCollection` provides a property to retrieve the `TSource` object that produced it, you do not need this overload of SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>). Instead, you can use SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>) and navigate back to the `TSource` object through the `TCollection` object.

In query expression syntax, each `from` clause (C#) or `From` clause (Visual Basic) after the initial one translates to an invocation of SelectMany.

See also

- from clause (C# Reference)
- From Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)

Source: SelectMany.cs

Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
```

```
Func<TSource,int,System.Collections.Generic.IEnumerable<TCollection>>
collectionSelector, Func<TSource,TCollection,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TCollection`

The type of the intermediate elements collected by `collectionSelector`.

`TResult`

The type of the elements of the resulting sequence.

Parameters

`source` IEnumerable<TSource>

A sequence of values to project.

`collectionSelector` Func<TSource,Int32,IEnumerable<TCollection>>

A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

`resultSelector` Func<TSource,TCollection,TResult>

A transform function to apply to each element of the intermediate sequence.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the one-to-many transform function `collectionSelector` on each element of `source` and then mapping each of those sequence elements and their corresponding source element to a result element.

Exceptions

ArgumentNullException

`source` or `collectionSelector` or `resultSelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>) method is useful when you have to keep the elements of `source` in scope for query logic that occurs after the call to SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>). See the Example section for a code example. If there is a bidirectional relationship between objects of type `TSource` and objects of type `TCollection`, that is, if an object of type `TCollection` provides a property to retrieve the `TSource` object that produced it, you do not need this overload of SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>). Instead, you can use SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>) and navigate back to the `TSource` object through the `TCollection` object.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SelectMany<TSource,TResult>(IEnumerable<TSource>,

Func<TSource,IEnumerable<TResult>>)

Source: SelectMany.cs

Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.

```
C#

public static System.Collections.Generic.IEnumerable<TResult>
SelectMany<TSource,TResult> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,System.Collections.Generic.IEnumerable<TResult>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the elements of the sequence returned by `selector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to project.

`selector` Func<TSource,IEnumerable<TResult>>

A transform function to apply to each element.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the one-to-many transform function on each element of the input sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>) to perform a one-to-many projection over an array.

C#

```
class PetOwner
{
    public string Name { get; set; }
    public List<String> Pets { get; set; }
}

public static void SelectManyEx1()
{
    PetOwner[] petOwners =
        { new PetOwner { Name="Higa, Sidney",
              Pets = new List<string>{ "Scruffy", "Sam" } },
          new PetOwner { Name="Ashkenazi, Ronen",
              Pets = new List<string>{ "Walker", "Sugar" } },
          new PetOwner { Name="Price, Vernette",
              Pets = new List<string>{ "Scratches", "Diesel" } } };

    // Query using SelectMany().
    IEnumerable<string> query1 = petOwners.SelectMany(petOwner =>
petOwner.Pets);

    Console.WriteLine("Using SelectMany():");

    // Only one foreach loop is required to iterate
    // through the results since it is a
    // one-dimensional collection.
    foreach (string pet in query1)
    {
        Console.WriteLine(pet);
    }

    // This code shows how to use Select()
    // instead of SelectMany().
    IEnumerable<List<String>> query2 =
        petOwners.Select(petOwner => petOwner.Pets);

    Console.WriteLine("\nUsing Select():");

    // Notice that two foreach loops are required to
    // iterate through the results
    // because the query returns a collection of arrays.
    foreach (List<String> petList in query2)
    {
        foreach (string pet in petList)
        {
            Console.WriteLine(pet);
```

```
        }
        Console.WriteLine();
    }
}

/*
 This code produces the following output:

 Using SelectMany():
 Scruffy
 Sam
 Walker
 Sugar
 Scratches
 Diesel

 Using Select():
 Scruffy
 Sam

 Walker
 Sugar

 Scratches
 Diesel
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>) method enumerates the input sequence, uses a transform function to map each element to an IEnumerable<T>, and then enumerates and yields the elements of each such IEnumerable<T> object. That is, for each element of `source`, `selector` is invoked and a sequence of values is returned. SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>) then flattens this two-dimensional collection of collections into a one-dimensional IEnumerable<T> and returns it. For example, if a query uses SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>) to obtain the orders (of type `Order`) for each customer in a database, the result is of type `IEnumerable<Order>` in C# or

`IEnumerable(Of Order)` in Visual Basic. If instead the query uses Select to obtain the orders, the collection of collections of orders is not combined and the result is of type `IEnumerable<List<Order>>` in C# or `IEnumerable(Of List(Of Order))` in Visual Basic.

In query expression syntax, each `from` clause (C#) or `From` clause (Visual Basic) after the initial one translates to an invocation of SelectMany.

See also

- from clause (C# Reference)
- From Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SelectMany<TSource,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)

Source: SelectMany.cs

Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
SelectMany<TSource,TResult> (this
```

```
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,int,System.Collections.Generic.IEnumerable<TResult>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the elements of the sequence returned by `selector`.

Parameters

`source` IEnumerable<TSource>

A sequence of values to project.

`selector` Func<TSource,Int32,IEnumerable<TResult>>

A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the one-to-many transform function on each element of an input sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>) to perform a one-to-many projection over an array and use the index of each outer element.

```
C#
```

```
class PetOwner
{
    public string Name { get; set; }
    public List<string> Pets { get; set; }
}

public static void SelectManyEx2()
{
    PetOwner[] petOwners =
        { new PetOwner { Name="Higa, Sidney",
              Pets = new List<string>{ "Scruffy", "Sam" } },
          new PetOwner { Name="Ashkenazi, Ronen",
              Pets = new List<string>{ "Walker", "Sugar" } },
          new PetOwner { Name="Price, Vernette",
              Pets = new List<string>{ "Scratches", "Diesel" } },
          new PetOwner { Name="Hines, Patrick",
              Pets = new List<string>{ "Dusty" } } };

    // Project the items in the array by appending the index
    // of each PetOwner to each pet's name in that petOwner's
    // array of pets.
    IEnumerable<string> query =
        petOwners.SelectMany((petOwner, index) =>
                                 petOwner.Pets.Select(pet => index +
pet));

    foreach (string pet in query)
    {
        Console.WriteLine(pet);
    }
}

// This code produces the following output:
//
// 0Scruffy
// 0Sam
// 1Walker
// 1Sugar
// 2Scratches
// 2Diesel
// 3Dusty
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>) method enumerates the input sequence, uses a transform function to map each element to an IEnumerable<T>, and then enumerates and yields the elements of each such IEnumerable<T> object. That is, for each element of `source`, `selector` is invoked and a sequence of values is returned. SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>) then flattens this two-dimensional collection of collections into a one-dimensional IEnumerable<T> and returns it. For example, if a query uses SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>) to obtain the orders (of type `Order`) for each customer in a database, the result is of type `IEnumerable<Order>` in C# or `IEnumerable(Of Order)` in Visual Basic. If instead the query uses Select to obtain the orders, the collection of collections of orders is not combined and the result is of type `IEnumerable<List<Order>>` in C# or `IEnumerable(Of List(Of Order))` in Visual Basic.

The first argument to `selector` represents the element to process. The second argument to `selector` represents the zero-based index of that element in the source sequence. This can be useful if the elements are in a known order and you want to do something with an element at a particular index, for example. It can also be useful if you want to retrieve the index of one or more elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.SequenceEqual Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Determines whether two sequences are equal according to an equality comparer.

Overloads

Expand table

SequenceEqual<TSource> (IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource> (IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.

SequenceEqual<TSource> (IEnumerable<TSource>, IEnumerable<TSource>)

Source: SequenceEqual.cs

Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.

C#

```
public static bool SequenceEqual<TSource> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> to compare to `second`.

`second` IEnumerable<TSource>

An IEnumerable<T> to compare to the first sequence.

Returns

Boolean

`true` if the two source sequences are of equal length and their corresponding elements are equal according to the default equality comparer for their type; otherwise, `false`.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following code examples demonstrate how to use `SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` to determine whether two sequences are equal. In the first two examples, the method determines whether the compared sequences contain references to the same objects. In the third and fourth examples, the method compares the actual data of the objects within the sequences.

In this example the sequences are equal.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void SequenceEqualEx1()
{
```

```
    Pet pet1 = new Pet { Name = "Turbo", Age = 2 };
    Pet pet2 = new Pet { Name = "Peanut", Age = 8 };

    // Create two lists of pets.
    List<Pet> pets1 = new List<Pet> { pet1, pet2 };
    List<Pet> pets2 = new List<Pet> { pet1, pet2 };

    bool equal = pets1.SequenceEqual(pets2);

    Console.WriteLine(
        "The lists {0} equal.",
        equal ? "are" : "are not");
}

/*
 This code produces the following output:

 The lists are equal.
*/
```

The following code example compares two sequences that are not equal. Note that the sequences contain identical data, but because the objects that they contain have different references, the sequences are not considered equal.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void SequenceEqualEx2()
{
    Pet pet1 = new Pet() { Name = "Turbo", Age = 2 };
    Pet pet2 = new Pet() { Name = "Peanut", Age = 8 };

    // Create two lists of pets.
    List<Pet> pets1 = new List<Pet> { pet1, pet2 };
    List<Pet> pets2 =
        new List<Pet> { new Pet { Name = "Turbo", Age = 2 },
                        new Pet { Name = "Peanut", Age = 8 } };

    bool equal = pets1.SequenceEqual(pets2);

    Console.WriteLine("The lists {0} equal.", equal ? "are" : "are not");
}

/*
 This code produces the following output:
```

```
 The lists are not equal.
*/
```

If you want to compare the actual data of the objects in the sequences instead of just comparing their references, you have to implement the IEqualityComparer<T> generic interface in your class. The following code example shows how to implement this interface in a helper class and provide GetHashCode and Equals methods.

C#

```
public class ProductA : IEquatable<ProductA>
{
    public string Name { get; set; }
    public int Code { get; set; }

    public bool Equals(ProductA other)
    {
        if (other is null)
            return false;

        return this.Name == other.Name && this.Code == other.Code;
    }

    public override bool Equals(object obj) => Equals(obj as ProductA);
    public override int GetHashCode() => (Name, Code).GetHashCode();
}
```

After you implement this interface, you can use sequences of `ProductA` objects in the `SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` method, as shown in the following example:

C#

```
ProductA[] storeA = { new ProductA { Name = "apple", Code = 9 },
                      new ProductA { Name = "orange", Code = 4 } };

ProductA[] storeB = { new ProductA { Name = "apple", Code = 9 },
                      new ProductA { Name = "orange", Code = 4 } };

bool equalAB = storeA.SequenceEqual(storeB);

Console.WriteLine("Equal? " + equalAB);

/*
    This code produces the following output:

    Equal? True
*/
```

Remarks

The `SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` method enumerates the two source sequences in parallel and compares corresponding elements by using the default equality comparer for `TSource`, Default.

The default equality comparer, Default, is used to compare values of the types. To compare a custom data type, you need to override the Equals and the GetHashCode methods, and optionally implement the IEquatable<T> generic interface in the custom type. For more information, see the Default property.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SequenceEqual<TSource> (IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: SequenceEqual.cs

Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.

C#

```
public static bool SequenceEqual<TSource> (this
System.Collections.Generic.IEnumerable<TSource> first,
```

```
System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> to compare to `second`.

`second` IEnumerable<TSource>

An IEnumerable<T> to compare to the first sequence.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to use to compare elements.

Returns

Boolean

`true` if the two source sequences are of equal length and their corresponding elements compare equal according to `comparer`; otherwise, `false`.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following example shows how to implement an equality comparer that can be used in the SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method.

C#

```
public class Product
{
    public string Name { get; set; }
    public int Code { get; set; }
```

```
}

// Custom comparer for the Product class
class ProductComparer : IEqualityComparer<Product>
{
    // Products are equal if their names and product numbers are equal.
    public bool Equals(Product x, Product y)
    {

        //Check whether the compared objects reference the same data.
        if (Object.ReferenceEquals(x, y)) return true;

        //Check whether any of the compared objects is null.
        if (Object.ReferenceEquals(x, null) || Object.ReferenceEquals(y,
null))
            return false;

        //Check whether the products' properties are equal.
        return x.Code == y.Code && x.Name == y.Name;
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public int GetHashCode(Product product)
    {
        //Check whether the object is null
        if (Object.ReferenceEquals(product, null)) return 0;

        //Get hash code for the Name field if it is not null.
        int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = product.Code.GetHashCode();

        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this comparer, you can use sequences of `Product` objects in the SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method, as shown in the following example:

C#

```
Product[] storeA = { new Product { Name = "apple", Code = 9 },
                     new Product { Name = "orange", Code = 4 } };

Product[] storeB = { new Product { Name = "apple", Code = 9 },
```

```
                        new Product { Name = "orange", Code = 4 } };

bool equalAB = storeA.SequenceEqual(storeB, new ProductComparer());

Console.WriteLine("Equal? " + equalAB);

/*
    This code produces the following output:

    Equal? True
*/
```

Remarks

The SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method enumerates the two source sequences in parallel and compares corresponding elements by using the specified IEqualityComparer<T>. If `comparer` is `null`, the default equality comparer, Default, is used to compare elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Single Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns a single, specific element of a sequence.

Overloads

Expand table

Single<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
Single<TSource> (IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.

Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Single.cs

Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

C#

```
public static TSource Single<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return a single element from.

`predicate` Func<TSource,Boolean>

A function to test an element for a condition.

Returns

TSource

The single element of the input sequence that satisfies a condition.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

No element satisfies the condition in `predicate`.

-or-

More than one element satisfies the condition in `predicate`.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use Single<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to select the only element of an array that satisfies a condition.

C#

```
string[] fruits = { "apple", "banana", "mango",
                    "orange", "passionfruit", "grape" };

string fruit1 = fruits.Single(fruit => fruit.Length > 10);

Console.WriteLine(fruit1);

/*
```

```
 This code produces the following output:

 passionfruit
*/
```

The following code example demonstrates that Single<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) throws an exception when the sequence does not contain exactly one element that satisfies the condition.

```C#
string fruit2 = null;

try
{
    fruit2 = fruits.Single(fruit => fruit.Length > 15);
}
catch (System.InvalidOperationException)
{
    Console.WriteLine(@"The collection does not contain exactly
                        one element whose length is greater than 15.");
}

Console.WriteLine(fruit2);

// This code produces the following output:
//
// The collection does not contain exactly
// one element whose length is greater than 15.
```

Remarks

The Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>) method throws an exception if the input sequence contains no matching element. To instead return `null` when no matching element is found, use SingleOrDefault.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Single<TSource>(IEnumerable<TSource>)

Source: Single.cs

Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.

```
C#

public static TSource Single<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return the single element of.

Returns

TSource

The single element of the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The input sequence contains more than one element.

-or-

The input sequence is empty.

Examples

The following code example demonstrates how to use Single<TSource> (IEnumerable<TSource>) to select the only element of an array.

C#

```
string[] fruits1 = { "orange" };

string fruit1 = fruits1.Single();

Console.WriteLine(fruit1);

/*
 This code produces the following output:

 orange
*/
```

The following code example demonstrates that Single<TSource> (IEnumerable<TSource>) throws an exception when the sequence does not contain exactly one element.

C#

```
string[] fruits2 = { "orange", "apple" };
string fruit2 = null;

try
{
    fruit2 = fruits2.Single();
}
catch (System.InvalidOperationException)
{
    Console.WriteLine("The collection does not contain exactly one
element.");
}

Console.WriteLine(fruit2);

/*
 This code produces the following output:

 The collection does not contain exactly one element.
*/
```

Remarks

The Single<TSource>(IEnumerable<TSource>) method throws an exception if the input sequence is empty. To instead return `null` when the input sequence is empty, use SingleOrDefault.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.SingleOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns a single, specific element of a sequence, or a default value if that element is not found.

Overloads

Expand table

SingleOrDefault<TSource> (IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource> (IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

SingleOrDefault<TSource> (IEnumerable<TSource>)

Source: Single.cs

Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

```
C#

public static TSource? SingleOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return the single element of.

Returns

TSource

The single element of the input sequence, or `default`(`TSource`) if the sequence contains no elements.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The input sequence contains more than one element.

Examples

The following code example demonstrates how to use SingleOrDefault<TSource> (IEnumerable<TSource>) to select the only element of an array.

```
C#

string[] fruits1 = { "orange" };

string fruit1 = fruits1.SingleOrDefault();
```

```
Console.WriteLine(fruit1);

/*
 This code produces the following output:

 orange
*/
```

The following code example demonstrates that SingleOrDefault<TSource>(IEnumerable<TSource>) returns a default value when the sequence is empty.

C#

```
string[] fruits2 = { };

string fruit2 = fruits2.SingleOrDefault();

Console.WriteLine(
    String.IsNullOrEmpty(fruit2) ? "No such string!" : fruit2);

/*
 This code produces the following output:

 No such string!
*/
```

Sometimes the value of `default(TSource)` is not the default value that you want to use if the collection contains no elements. Instead of checking the result for the unwanted default value and then changing it if necessary, you can use the DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) method to specify the default value that you want to use if the collection is empty. Then, call Single<TSource>(IEnumerable<TSource>) to obtain the element. The following code example uses both techniques to obtain a default value of 1 if a collection of page numbers is empty. Because the default value for an integer is 0, which is not usually a valid page number, the default value must be specified as 1 instead. The first result variable is checked for the unwanted default value after the query has finished executing. The second result variable is obtained by using DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) to specify a default value of 1.

C#

```
int[] pageNumbers = { };

// Setting the default value to 1 after the query.
int pageNumber1 = pageNumbers.SingleOrDefault();
if (pageNumber1 == 0)
```

```
{
    pageNumber1 = 1;
}
Console.WriteLine("The value of the pageNumber1 variable is {0}",
pageNumber1);

// Setting the default value to 1 by using DefaultIfEmpty() in the query.
int pageNumber2 = pageNumbers.DefaultIfEmpty(1).Single();
Console.WriteLine("The value of the pageNumber2 variable is {0}",
pageNumber2);

/*
 This code produces the following output:

 The value of the pageNumber1 variable is 1
 The value of the pageNumber2 variable is 1
*/
```

Remarks

The default value for reference and nullable types is `null`.

The SingleOrDefault method does not provide a way to specify a default value. If you want to specify a default value other than `default(TSource)`, use the DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource) method as described in the Example section.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SingleOrDefault<TSource>(IEnumerable<TSource>,

Func<TSource,Boolean>)

Source: Single.cs

Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

```
C#

public static TSource? SingleOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return a single element from.

`predicate` Func<TSource,Boolean>

A function to test an element for a condition.

Returns

TSource

The single element of the input sequence that satisfies the condition, or `default`(`TSource`) if no such element is found.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

More than one element satisfies the condition in `predicate`.

Examples

The following code example demonstrates how to use SingleOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to select the only element of an array that satisfies a condition.

C#

```
string[] fruits = { "apple", "banana", "mango",
                    "orange", "passionfruit", "grape" };

string fruit1 = fruits.SingleOrDefault(fruit => fruit.Length > 10);

Console.WriteLine(fruit1);

/*
 This code produces the following output:

 passionfruit
*/
```

The following code example demonstrates that SingleOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) returns a default value when the sequence contains no elements that satisfy the condition.

C#

```
string fruit2 =
    fruits.SingleOrDefault(fruit => fruit.Length > 15);

Console.WriteLine(
    String.IsNullOrEmpty(fruit2) ? "No such string!" : fruit2);

/*
 This code produces the following output:

 No such string!
*/
```

Remarks

The default value for reference and nullable types is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SingleOrDefault<TSource> (IEnumerable<TSource>, TSource)

Source: Single.cs

Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

C#

```
public static TSource SingleOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return the single element of.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

The single element of the input sequence, or `defaultValue` if the sequence contains no elements.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The input sequence contains more than one element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

SingleOrDefault<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>, TSource)

Source: Single.cs

Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

C#

```
public static TSource SingleOrDefault<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,bool> predicate, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return a single element from.

`predicate` Func<TSource,Boolean>

A function to test an element for a condition.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

The single element of the input sequence that satisfies the condition, or `defaultValue` if no such element is found.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

More than one element satisfies the condition in `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.Skip<TSource> (IEnumerable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Skip.cs

Bypasses a specified number of elements in a sequence and then returns the remaining elements.

C#

```
public static System.Collections.Generic.IEnumerable<TSource> Skip<TSource>
(this System.Collections.Generic.IEnumerable<TSource> source, int count);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return elements from.

`count` Int32

The number of elements to skip before returning the remaining elements.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements that occur after the specified index in the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Skip to skip a specified number of elements in an array and return the remaining elements.

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

Console.WriteLine("All grades except the first three:");
foreach (int grade in grades.Skip(3))
{
    Console.WriteLine(grade);
}

/*
 This code produces the following output:

All grades except the first three:
 56
 92
 98
 85
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

If `source` contains fewer than `count` elements, an empty IEnumerable<T> is returned. If `count` is less than or equal to zero, all elements of `source` are yielded.

The Take and Skip methods are functional complements. Given a collection sequence `coll` and an integer `n`, concatenating the results of `coll.Take(n)` and `coll.Skip(n)` yields the same sequence as `coll`.

In Visual Basic query expression syntax, a `Skip` clause translates to an invocation of Skip.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Skip Clause (Visual Basic)

Enumerable.SkipLast<TSource> (IEnumerable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Skip.cs

Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
SkipLast<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, int count);
```

Type Parameters

`TSource`

The type of the elements in the enumerable collection.

Parameters

`source` IEnumerable<TSource>

An enumerable collection instance.

`count` Int32

The number of elements to omit from the end of the collection.

Returns

IEnumerable<TSource>

A new enumerable collection that contains the elements from `source` minus `count` elements from the end of the collection.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

If `count` is not a positive number, this method returns an identical copy of the `source` enumerable collection.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Enumerable.SkipWhile Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.

Overloads

Expand table

SkipWhile<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource> (IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Skip.cs

Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
SkipWhile<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return elements from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use SkipWhile<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to skip elements of an array as long as a condition is true.

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

IEnumerable<int> lowerGrades =
    grades
    .OrderByDescending(grade => grade)
    .SkipWhile(grade => grade >= 80);

Console.WriteLine("All grades below 80:");
foreach (int grade in lowerGrades)
{
    Console.WriteLine(grade);
}
```

```
/*
 This code produces the following output:

 All grades below 80:
 70
 59
 56
*/
```

Remarks

The SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>) method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

This method tests each element of `source` by using `predicate` and skips the element if the result is `true`. After the predicate function returns `false` for an element, that element and the remaining elements in `source` are yielded and there are no more invocations of `predicate`.

If `predicate` returns `true` for all elements in the sequence, an empty IEnumerable<T> is returned.

The TakeWhile and SkipWhile methods are functional complements. Given a collection sequence `coll` and a pure function `p`, concatenating the results of `coll.TakeWhile(p)` and `coll.SkipWhile(p)` yields the same sequence as `coll`.

In Visual Basic query expression syntax, a `Skip While` clause translates to an invocation of SkipWhile.

See also

- Skip While Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)

Source: Skip.cs

Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
SkipWhile<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, Func<TSource,int,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to return elements from.

`predicate` Func<TSource,Int32,Boolean>

A function to test each source element for a condition; the second parameter of the function represents the index of the source element.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use SkipWhile<TSource> (IEnumerable<TSource>, Func<TSource,Int32,Boolean>) to skip elements of an array as long as a condition that depends on the element's index is true.

C#

```
int[] amounts = { 5000, 2500, 9000, 8000,
                    6500, 4000, 1500, 5500 };

IEnumerable<int> query =
    amounts.SkipWhile((amount, index) => amount > index * 1000);

foreach (int amount in query)
{
    Console.WriteLine(amount);
}

/*
 This code produces the following output:

 4000
 1500
 5500
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>) method tests each element of `source` by using `predicate` and skips the element if the result is `true`. After the predicate function returns `false` for an element, that element and the remaining elements in `source` are yielded and there are no more invocations of `predicate`.

If `predicate` returns `true` for all elements in the sequence, an empty IEnumerable<T> is returned.

The first argument of `predicate` represents the element to test. The second argument represents the zero-based index of the element within `source`.

The TakeWhile and SkipWhile methods are functional complements. Given a collection sequence `coll` and a pure function `p`, concatenating the results of `coll.TakeWhile(p)` and `coll.SkipWhile(p)` yields the same sequence as `coll`.

In Visual Basic query expression syntax, a `Skip While` clause translates to an invocation of SkipWhile.

See also

- Skip While Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Sum Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Computes the sum of a sequence of numeric values.

Overloads

Expand table

Sum(IEnumerable<Single>)	Computes the sum of a sequence of Single values.
Sum(IEnumerable<Nullable<Int32>>)	Computes the sum of a sequence of nullable Int32 values.
Sum(IEnumerable<Nullable<Single>>)	Computes the sum of a sequence of nullable Single values.
Sum(IEnumerable<Nullable<Int64>>)	Computes the sum of a sequence of nullable Int64 values.
Sum(IEnumerable<Nullable<Double>>)	Computes the sum of a sequence of nullable Double values.
Sum(IEnumerable<Int32>)	Computes the sum of a sequence of Int32 values.
Sum(IEnumerable<Int64>)	Computes the sum of a sequence of Int64 values.
Sum(IEnumerable<Double>)	Computes the sum of a sequence of Double values.
Sum(IEnumerable<Decimal>)	Computes the sum of a sequence of Decimal values.
Sum(IEnumerable<Nullable<Decimal>>)	Computes the sum of a sequence of nullable Decimal values.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.

Sum(IEnumerable<Single>)

Source: Sum.cs

Computes the sum of a sequence of Single values.

```
C#

public static float Sum (this
System.Collections.Generic.IEnumerable<float> source);
```

Parameters

`source` IEnumerable<Single>

A sequence of Single values to calculate the sum of.

Returns

Single

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Sum(IEnumerable<Single>) to sum the values of a sequence.

```
C#

List<float> numbers = new List<float> { 43.68F, 1.25F, 583.7F, 6.5F };

float sum = numbers.Sum();

Console.WriteLine("The sum of the numbers is {0}.", sum);

/*
 This code produces the following output:
```

```
 The sum of the numbers is 635.13.
*/
```

Remarks

This method returns zero if `source` contains no elements.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Nullable<Int32>>)

Source: Sum.cs

Computes the sum of a sequence of nullable Int32 values.

C#

```
public static int? Sum (this System.Collections.Generic.IEnumerable<int?>
source);
```

Parameters

`source` IEnumerable<Nullable<Int32>>

A sequence of nullable Int32 values to calculate the sum of.

Returns

Nullable<Int32>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Sum(IEnumerable<Nullable<Single>>)

Source: Sum.cs

Computes the sum of a sequence of nullable Single values.

```
C#

public static float? Sum (this
System.Collections.Generic.IEnumerable<float?> source);
```

Parameters

`source` IEnumerable<Nullable<Single>>

A sequence of nullable Single values to calculate the sum of.

Returns

Nullable<Single>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Sum(IEnumerable<Nullable<Single>>) to sum the values of a sequence.

```
C#

float?[] points = { null, 0, 92.83F, null, 100.0F, 37.46F, 81.1F };

float? sum = points.Sum();

Console.WriteLine("Total points earned: {0}", sum);
```

```
/*
 This code produces the following output:

 Total points earned: 311.39
*/
```

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Nullable<Int64>>)

Source: Sum.cs

Computes the sum of a sequence of nullable Int64 values.

C#

```
public static long? Sum (this
```

```
System.Collections.Generic.IEnumerable<long?> source);
```

Parameters

source IEnumerable<Nullable<Int64>>

A sequence of nullable Int64 values to calculate the sum of.

Returns

Nullable<Int64>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Nullable<Double>>)

Source: Sum.cs

Computes the sum of a sequence of nullable Double values.

C#

```
public static double? Sum (this
System.Collections.Generic.IEnumerable<double?> source);
```

Parameters

`source` IEnumerable<Nullable<Double>>

A sequence of nullable Double values to calculate the sum of.

Returns

Nullable<Double>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Double.MaxValue.

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Int32>)

Source: Sum.cs

Computes the sum of a sequence of Int32 values.

C#

```
public static int Sum (this System.Collections.Generic.IEnumerable<int>
source);
```

Parameters

`source` IEnumerable<Int32>

A sequence of Int32 values to calculate the sum of.

Returns

Int32

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Remarks

This method returns zero if `source` contains no elements.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Int64>)

Source: Sum.cs

Computes the sum of a sequence of Int64 values.

```
C#
```

```
public static long Sum (this System.Collections.Generic.IEnumerable<long>
source);
```

Parameters

`source` IEnumerable<Int64>

A sequence of Int64 values to calculate the sum of.

Returns

Int64

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Remarks

This method returns zero if `source` contains no elements.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Double>)

Source: Sum.cs

Computes the sum of a sequence of Double values.

C#

```
public static double Sum (this
System.Collections.Generic.IEnumerable<double> source);
```

Parameters

`source` IEnumerable<Double>

A sequence of Double values to calculate the sum of.

Returns

Double

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Double.MaxValue.

Remarks

This method returns zero if `source` contains no elements.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Decimal>)

Source: Sum.cs

Computes the sum of a sequence of Decimal values.

C#

```
public static decimal Sum (this
System.Collections.Generic.IEnumerable<decimal> source);
```

Parameters

`source` IEnumerable<Decimal>

A sequence of Decimal values to calculate the sum of.

Returns

Decimal

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Remarks

The Sum(IEnumerable<Decimal>) method returns zero if `source` contains no elements.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum(IEnumerable<Nullable<Decimal>>)

Source: Sum.cs

Computes the sum of a sequence of nullable Decimal values.

```
C#
```

```
public static decimal? Sum (this
System.Collections.Generic.IEnumerable<decimal?> source);
```

Parameters

`source` IEnumerable<Nullable<Decimal>>

A sequence of nullable Decimal values to calculate the sum of.

Returns

Nullable<Decimal>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)

Source: Sum.cs

Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static float Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:
```

```
 The total weight of the packages is: 83.7
*/
```

Remarks

The Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>) method returns zero if `source` contains no elements.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Single.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)

Source: Sum.cs

Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.

```C#
public static long? Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Int64>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

ⓘ **Note**

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Int64>` in C# or `Nullable(Of Int64)` in Visual Basic

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)

Source: Sum.cs

Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static int? Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int?
> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Int32>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
```

```
                new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Int32>` in C# or `Nullable(Of Int32)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)

Source: Sum.cs

Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double? Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Double.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Double>` in C# or `Nullable(Of Double)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)

Source: Sum.cs

Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static float? Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,float?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
```

```
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Single>` in C# or `Nullable(Of Single)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)

Source: Sum.cs

Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static long Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Int64

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:
```

```
 The total weight of the packages is: 83.7
*/
```

Remarks

This method returns zero if `source` contains no elements.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Int64.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)

Source: Sum.cs

Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.

```
C#
```

```
public static int Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, Func<TSource,int>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Int32

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the

overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

This method returns zero if `source` contains no elements.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Int32.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)

Source: Sum.cs

Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static double Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,double> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Double.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:
```

```
 The total weight of the packages is: 83.7
*/
```

Remarks

This method returns zero if `source` contains no elements.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Double.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)

Source: Sum.cs

Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.

```
C#
```

```
public static decimal Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> ⓘ **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the

overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

This method returns zero if `source` contains no elements.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically Decimal.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)

Source: Sum.cs

Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static decimal? Sum<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,decimal?> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence of values that are used to calculate a sum.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    double totalWeight = packages.Sum(pkg => pkg.Weight);
```

```
    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
 This code produces the following output:

 The total weight of the packages is: 83.7
*/
```

Remarks

Items in `source` that are `null` are excluded from the computation of the sum. This method returns zero if `source` contains no elements or all elements are `null`.

You can apply this method to a sequence of arbitrary values if you provide a function, `selector`, that projects the members of `source` into a numeric type, specifically `Nullable<Decimal>` in C# or `Nullable(Of Decimal)` in Visual Basic.

In Visual Basic query expression syntax, an `Aggregate Into Sum()` clause translates to an invocation of Sum.

See also

- Aggregate Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Take Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

Take<TSource> (IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource> (IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.

Take<TSource>(IEnumerable<TSource>, Int32)

Source: Take.cs

Returns a specified number of contiguous elements from the start of a sequence.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Take<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, int count);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

The sequence to return elements from.

`count` Int32

The number of elements to return.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the specified number of elements from the start of the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Take to return elements from the start of a sequence.

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

IEnumerable<int> topThreeGrades =
    grades.OrderByDescending(grade => grade).Take(3);

Console.WriteLine("The top three grades are:");
foreach (int grade in topThreeGrades)
{
    Console.WriteLine(grade);
}
/*
 This code produces the following output:

 The top three grades are:
 98
 92
 85
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the

action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

Take enumerates `source` and yields elements until `count` elements have been yielded or `source` contains no more elements. If `count` exceeds the number of elements in `source`, all elements of `source` are returned.

If `count` is less than or equal to zero, `source` is not enumerated and an empty IEnumerable<T> is returned.

The Take and Skip methods are functional complements. Given a collection sequence `coll` and an integer `n`, concatenating the results of `coll.Take(n)` and `coll.Skip(n)` yields the same sequence as `coll`.

In Visual Basic query expression syntax, a `Take` clause translates to an invocation of Take.

See also

- Take Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Take<TSource>(IEnumerable<TSource>, Range)

Source: Take.cs

Returns a specified range of contiguous elements from a sequence.

```
C#

public static System.Collections.Generic.IEnumerable<TSource>
Take<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, Range range);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to return elements from.

`range` Range

The range of elements to return, which has start and end indexes either from the beginning or the end of the sequence.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the specified range of elements from the `source` sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

Take enumerates `source` and yields elements whose indices belong to the specified `range`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.TakeLast<TSource> (IEnumerable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Take.cs

Returns a new enumerable collection that contains the last `count` elements from `source`.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
TakeLast<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, int count);
```

Type Parameters

`TSource`

The type of the elements in the enumerable collection.

Parameters

`source` IEnumerable<TSource>

An enumerable collection instance.

`count` Int32

The number of elements to take from the end of the collection.

Returns

IEnumerable<TSource>

A new enumerable collection that contains the last `count` elements from `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

If `count` is not a positive number, this method returns an empty enumerable collection.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Enumerable.TakeWhile Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Returns elements from a sequence as long as a specified condition is true, and then skips the remaining elements.

Overloads

Expand table

TakeWhile<TSource> (IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
TakeWhile<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.

TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)

Source: Take.cs

Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
TakeWhile<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, Func<TSource,int,bool> predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

The sequence to return elements from.

`predicate` Func<TSource,Int32,Boolean>

A function to test each source element for a condition; the second parameter of the function represents the index of the source element.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains elements from the input sequence that occur before the element at which the test no longer passes.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use TakeWhile<TSource> (IEnumerable<TSource>, Func<TSource,Int32,Boolean>) to return elements from the start of a sequence as long as a condition that uses the element's index is true.

C#

```
string[] fruits = { "apple", "passionfruit", "banana", "mango",
                    "orange", "blueberry", "grape", "strawberry" };

IEnumerable<string> query =
    fruits.TakeWhile((fruit, index) => fruit.Length >= index);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}

/*
 This code produces the following output:

 apple
```

```
  passionfruit
  banana
  mango
  orange
  blueberry
 */
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>) method tests each element of `source` by using `predicate` and yields the element if the result is `true`. Enumeration stops when the predicate function returns `false` for an element or when `source` contains no more elements.

The first argument of `predicate` represents the element to test. The second argument represents the zero-based index of the element within `source`.

The TakeWhile and SkipWhile methods are functional complements. Given a collection sequence `coll` and a pure function `p`, concatenating the results of `coll.TakeWhile(p)` and `coll.SkipWhile(p)` yields the same sequence as `coll`.

In Visual Basic query expression syntax, a `Take While` clause translates to an invocation of TakeWhile.

See also

- Take While Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Take.cs

Returns elements from a sequence as long as a specified condition is true.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
TakeWhile<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence to return elements from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from the input sequence that occur before the element at which the test no longer passes.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use TakeWhile<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to return elements from the start of a sequence as long as a condition is true.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange",
                      "passionfruit", "grape" };

IEnumerable<string> query =
    fruits.TakeWhile(fruit => String.Compare("orange", fruit, true) !=
0);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}

/*
 This code produces the following output:

 apple
 banana
 mango
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>) method tests each element of `source` by using `predicate` and yields the element if the result is `true`. Enumeration stops when the predicate function returns `false` for an element or when `source` contains no more elements.

The TakeWhile and SkipWhile methods are functional complements. Given a collection sequence `coll` and a pure function `p`, concatenating the results of `coll.TakeWhile(p)` and `coll.SkipWhile(p)` yields the same sequence as `coll`.

In Visual Basic query expression syntax, a `Take While` clause translates to an invocation of TakeWhile.

See also

- Take While Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ThenBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Performs a subsequent ordering of the elements in a sequence in ascending order.

Overloads

Expand table

ThenBy<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

ThenBy<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>)

Source: OrderBy.cs

Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
ThenBy<TSource,TKey> (this System.Linq.IOrderedEnumerable<TSource>
source, Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use ThenBy<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>) to perform a secondary ordering of the elements in a sequence.

C#

```
string[] fruits = { "grape", "passionfruit", "banana", "mango",
                      "orange", "raspberry", "apple", "blueberry" };

// Sort the strings first by their length and then
//alphabetically by passing the identity selector function.
IEnumerable<string> query =
    fruits.OrderBy(fruit => fruit.Length).ThenBy(fruit => fruit);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}
```

```
/*
    This code produces the following output:

    apple
    grape
    mango
    banana
    orange
    blueberry
    raspberry
    passionfruit
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

ThenBy and ThenByDescending are defined to extend the type IOrderedEnumerable<TElement>, which is also the return type of these methods. This design enables you to specify multiple sort criteria by applying any number of ThenBy or ThenByDescending methods.

> ⓘ **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

This method compares keys by using the default comparer Default.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

In query expression syntax, an `orderby [first criterion], [second criterion]` (C#) or `Order By [first criterion], [second criterion]` (Visual Basic) clause translates to an invocation of ThenBy.

See also

- orderby clause (C# Reference)
- Order By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ThenBy<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: OrderBy.cs

Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
ThenBy<TSource,TKey> (this System.Linq.IOrderedEnumerable<TSource>
source, Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

ThenBy and ThenByDescending are defined to extend the type IOrderedEnumerable<TElement>, which is also the return type of these methods.

This design enables you to specify multiple sort criteria by applying any number of ThenBy or ThenByDescending methods.

> ⓘ **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

If `comparer` is `null`, the default comparer Default is used to compare keys.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ThenByDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Performs a subsequent ordering of the elements in a sequence in descending order.

Overloads

Expand table

ThenByDescending<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

ThenByDescending<TSource,TKey> (IOrderedEnumerable<TSource>, Func<TSource,TKey>)

Source: OrderBy.cs

Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.

C#

```
public static System.Linq.IOrderedEnumerable<TSource>
ThenByDescending<TSource,TKey> (this
System.Linq.IOrderedEnumerable<TSource> source, Func<TSource,TKey>
keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

ThenBy and ThenByDescending are defined to extend the type IOrderedEnumerable<TElement>, which is also the return type of these methods. This design enables you to specify multiple sort criteria by applying any number of ThenBy or ThenByDescending methods.

> ⓘ **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

This sorting method compares keys by using the default comparer Default.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

In C# query expression syntax, an `orderby [first criterion], [second criterion] descending` clause translates to an invocation of ThenByDescending.

In Visual Basic query expression syntax, an `Order By [first criterion], [second criterion] Descending` clause translates to an invocation of ThenByDescending.

See also

- orderby clause (C# Reference)
- Order By Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ThenByDescending<TSource,TKey> (IOrderedEnumerable<TSource>,

Func<TSource,TKey>, IComparer<TKey>)

Source: OrderBy.cs

Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

```
C#

public static System.Linq.IOrderedEnumerable<TSource>
ThenByDescending<TSource,TKey> (this
System.Linq.IOrderedEnumerable<TSource> source, Func<TSource,TKey>
keySelector, System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedEnumerable<TSource>

An IOrderedEnumerable<TElement> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use ThenByDescending<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>) to perform a secondary ordering of the elements in a sequence in descending order by using a custom comparer.

C#

```
public class CaseInsensitiveComparer : IComparer<string>
{
    public int Compare(string x, string y)
    {
        return string.Compare(x, y, true);
    }
}

public static void ThenByDescendingEx1()
{
    string[] fruits = { "apPLe", "baNanA", "apple", "APple", "orange",
"BAnana", "ORANGE", "apPLE" };

    // Sort the strings first ascending by their length and
    // then descending using a custom case insensitive comparer.
    IEnumerable<string> query =
        fruits
        .OrderBy(fruit => fruit.Length)
        .ThenByDescending(fruit => fruit, new CaseInsensitiveComparer());

    foreach (string fruit in query)
    {
        Console.WriteLine(fruit);
    }
}

/*
    This code produces the following output:

    apPLe
    apple
    APple
    apPLE
    orange
    ORANGE
    baNanA
    BAnana
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

To order a sequence by the values of the elements themselves, specify the identity function (`x => x` in C# or `Function(x) x` in Visual Basic) for `keySelector`.

ThenBy and ThenByDescending are defined to extend the type IOrderedEnumerable<TElement>, which is also the return type of these methods. This design enables you to specify multiple sort criteria by applying any number of ThenBy or ThenByDescending methods.

> ⓘ **Note**
>
> Because IOrderedEnumerable<TElement> inherits from IEnumerable<T>, you can call OrderBy or OrderByDescending on the results of a call to OrderBy, OrderByDescending, ThenBy or ThenByDescending. Doing this introduces a new primary ordering that ignores the previously established ordering.

If `comparer` is `null`, the default comparer Default is used to compare keys.

This method performs a stable sort; that is, if the keys of two elements are equal, the order of the elements is preserved. In contrast, an unstable sort does not preserve the order of elements that have the same key.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Enumerable.ToArray<TSource> (IEnumerable<TSource>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: ToCollection.cs

Creates an array from a IEnumerable<T>.

C#

```
public static TSource[] ToArray<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

An IEnumerable<T> to create an array from.

Returns

TSource[]

An array that contains the elements from the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use ToArray to force immediate query evaluation and return an array of results.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void ToArrayEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7 },
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 } };

    string[] companies = packages.Select(pkg => pkg.Company).ToArray();

    foreach (string company in companies)
    {
        Console.WriteLine(company);
    }
}

/*
 This code produces the following output:

 Coho Vineyard
 Lucerne Publishing
 Wingtip Toys
 Adventure Works
*/
```

Remarks

The ToArray<TSource>(IEnumerable<TSource>) method forces immediate query evaluation and returns an array that contains the query results. You can append this method to your query in order to obtain a cached copy of the query results.

ToList has similar behavior but returns a List<T> instead of an array.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ToDictionary Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Creates a Dictionary<TKey,TValue> from an IEnumerable<T>.

Overloads

Expand table

ToDictionary<TKey,TValue>(IEnumerable<ValueTuple<TKey,TValue>>, IEqualityComparer<TKey>)	Creates a dictionary from an enumeration according to specified key equality comparer.
ToDictionary<TKey,TValue>(IEnumerable<KeyValuePair<TKey,TValue>>, IEqualityComparer<TKey>)	Creates a dictionary from an enumeration according to specified key comparer.
ToDictionary<TKey,TValue>(IEnumerable<KeyValuePair<TKey,TValue>>)	Creates a dictionary from an enumeration according to the default comparer for the key type.
ToDictionary<TKey,TValue>(IEnumerable<ValueTuple<TKey,TValue>>)	Creates a dictionary from an enumeration according to the default comparer for the key type.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.

ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.

ToDictionary<TKey,TValue>(IEnumerable<ValueTuple<TKey,TValue>>, IEqualityComparer<TKey>)

Source: ToCollection.cs

Creates a dictionary from an enumeration according to specified key equality comparer.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TValue>
ToDictionary<TKey,TValue> (this
System.Collections.Generic.IEnumerable<(TKey Key, TValue Value)> source,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TKey`

The type of the keys from elements of `source`.

`TValue`

The type of the values from elements of `source`.

Parameters

`source` IEnumerable<ValueTuple<TKey,TValue>>

The enumeration to create a dictionary from.

`comparer` IEqualityComparer<TKey>

An equality comparer to compare keys.

Returns

Dictionary<TKey,TValue>

A dictionary that contains keys and values from `source`.

Exceptions

ArgumentNullException

`source` is a `null` reference.

ArgumentException

`source` contains one or more duplicate keys.

Remarks

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ToDictionary<TKey,TValue> (IEnumerable<KeyValuePair<TKey,TValue>>, IEqualityComparer<TKey>)

Source: ToCollection.cs

Creates a dictionary from an enumeration according to specified key comparer.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TValue>
ToDictionary<TKey,TValue> (this
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValu
ePair<TKey,TValue>> source,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

TKey

The type of the keys from elements of `source`.

`TValue`

The type of the values from elements of `source`.

Parameters

`source` IEnumerable<KeyValuePair<TKey,TValue>>

The enumeration to create a dictionary from.

`comparer` IEqualityComparer<TKey>

An equality comparer to compare keys.

Returns

Dictionary<TKey,TValue>

A dictionary that contains keys and values from `source`.

Exceptions

ArgumentNullException

`source` is a `null` reference.

ArgumentException

`source` contains one or more duplicate keys.

Remarks

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ToDictionary<TKey,TValue> (IEnumerable<KeyValuePair<TKey,TValue>>)

Source: ToCollection.cs

Creates a dictionary from an enumeration according to the default comparer for the key type.

```
C#

public static System.Collections.Generic.Dictionary<TKey,TValue>
ToDictionary<TKey,TValue> (this
System.Collections.Generic.IEnumerable<System.Collections.Generic.KeyValu
ePair<TKey,TValue>> source);
```

Type Parameters

`TKey`

The type of the keys from elements of `source`.

`TValue`

The type of the values from elements of `source`.

Parameters

`source` IEnumerable<KeyValuePair<TKey,TValue>>

The enumeration to create a dictionary from.

Returns

Dictionary<TKey,TValue>

A dictionary that contains keys and values from `source` and uses the default comparer for the key type.

Exceptions

ArgumentNullException

`source` is a `null` reference.

ArgumentException

`source` contains one or more duplicate keys.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ToDictionary<TKey,TValue> (IEnumerable<ValueTuple<TKey,TValue>>)

Source: ToCollection.cs

Creates a dictionary from an enumeration according to the default comparer for the key type.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TValue>
ToDictionary<TKey,TValue> (this
System.Collections.Generic.IEnumerable<(TKey Key, TValue Value)> source);
```

Type Parameters

`TKey`

The type of the keys from elements of `source`.

`TValue`

The type of the values from elements of `source`.

Parameters

`source` IEnumerable<ValueTuple<TKey,TValue>>

The enumeration to create a dictionary from.

Returns

Dictionary<TKey,TValue>

A dictionary that contains keys and values from `source` and uses default comparer for the key type.

Exceptions

ArgumentNullException

`source` is a `null` reference.

ArgumentException

`source` contains one or more duplicate keys.

Applies to

▼ .NET 9 and .NET 8

Product	Versions
.NET	8, 9

ToDictionary<TSource,TKey,TElement> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)

Source: ToCollection.cs

Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TSource,TKey,TElement> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

Returns

Dictionary<TKey,TElement>

A Dictionary<TKey,TValue> that contains values of type `TElement` selected from the input sequence.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

-or-

`keySelector` produces a key that is `null`.

ArgumentException

`keySelector` produces duplicate keys for two elements.

Remarks

The ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>) method uses the default equality comparer Default to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ToDictionary<TSource,TKey,TElement> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)

Source: ToCollection.cs

Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TSource,TKey,TElement> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

Dictionary<TKey,TElement>

A Dictionary<TKey,TValue> that contains values of type `TElement` selected from the input sequence.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

-or-

`keySelector` produces a key that is `null`.

ArgumentException

`keySelector` produces duplicate keys for two elements.

Remarks

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ToDictionary<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)

Source: ToCollection.cs

Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TSource>
ToDictionary<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

Returns

Dictionary<TKey,TSource>

A Dictionary<TKey,TValue> that contains keys and values. The values within each group are in the same order as in `source`.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

-or-

`keySelector` produces a key that is `null`.

ArgumentException

`keySelector` produces duplicate keys for two elements.

Examples

The following code example demonstrates how to use ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) to create a Dictionary<TKey,TValue> by using a key selector.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
    public long TrackingNumber { get; set; }
}

public static void ToDictionaryEx1()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2,
TrackingNumber = 89453312L },
              new Package { Company = "Lucerne Publishing", Weight =
18.7, TrackingNumber = 89112755L },
              new Package { Company = "Wingtip Toys", Weight = 6.0,
TrackingNumber = 299456122L },
              new Package { Company = "Adventure Works", Weight = 33.8,
TrackingNumber = 4665518773L } };

    // Create a Dictionary of Package objects,
```

```
    // using TrackingNumber as the key.
    Dictionary<long, Package> dictionary =
        packages.ToDictionary(p => p.TrackingNumber);

    foreach (KeyValuePair<long, Package> kvp in dictionary)
    {
        Console.WriteLine(
            "Key {0}: {1}, {2} pounds",
            kvp.Key,
            kvp.Value.Company,
            kvp.Value.Weight);
    }
}

/*
 This code produces the following output:

 Key 89453312: Coho Vineyard, 25.2 pounds
 Key 89112755: Lucerne Publishing, 18.7 pounds
 Key 299456122: Wingtip Toys, 6 pounds
 Key 4665518773: Adventure Works, 33.8 pounds
*/
```

Remarks

The ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) method uses the default equality comparer Default to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>,

IEqualityComparer<TKey>)

Source: ToCollection.cs

Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.

```
C#

public static System.Collections.Generic.Dictionary<TKey,TSource>
ToDictionary<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the keys returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

Dictionary<TKey,TSource>

A Dictionary<TKey,TValue> that contains keys and values. The values within each group are in the same order as in `source`.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

-or-

`keySelector` produces a key that is `null`.

ArgumentException

`keySelector` produces duplicate keys for two elements.

Remarks

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ToHashSet Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.

ToHashSet<TSource>(IEnumerable<TSource>)

Source: ToCollection.cs

Creates a HashSet<T> from an IEnumerable<T>.

C#

```
public static System.Collections.Generic.HashSet<TSource>
ToHashSet<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to create a HashSet<T> from.

Returns

HashSet<TSource>

A HashSet<T> that contains values of type `TSource` selected from the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

See also

- ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.2, 4.8, 4.8.1
.NET Standard	2.1

ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: ToCollection.cs

Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.

C#

```
public static System.Collections.Generic.HashSet<TSource>
ToHashSet<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to create a HashSet<T> from.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare keys.

Returns

HashSet<TSource>

A HashSet<T> that contains values of type `TSource` selected from the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

See also

- ToHashSet<TSource>(IEnumerable<TSource>)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.7.2, 4.8, 4.8.1
.NET Standard	2.1

Enumerable.ToList<TSource> (IEnumerable<TSource>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: ToCollection.cs

Creates a List<T> from an IEnumerable<T>.

C#

```
public static System.Collections.Generic.List<TSource> ToList<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IEnumerable<TSource>

The IEnumerable<T> to create a List<T> from.

Returns

List<TSource>

A List<T> that contains elements from the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use ToList to force immediate query evaluation and return a List<T> that contains the query results.

C#

```
string[] fruits = { "apple", "passionfruit", "banana", "mango",
                    "orange", "blueberry", "grape", "strawberry" };

List<int> lengths = fruits.Select(fruit => fruit.Length).ToList();

foreach (int length in lengths)
{
    Console.WriteLine(length);
}

/*
 This code produces the following output:

 5
 12
 6
 5
 6
 9
 5
 10
*/
```

Remarks

The ToList<TSource>(IEnumerable<TSource>) method forces immediate query evaluation and returns a List<T> that contains the query results. You can append this method to your query in order to obtain a cached copy of the query results.

ToArray has similar behavior but returns an array instead of a List<T>.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.ToLookup Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Creates a generic Lookup<TKey,TElement> from an IEnumerable<T>.

Overloads

Expand table

ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.

ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)

Source: Lookup.cs

Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.

C#

```
public static System.Linq.ILookup<TKey,TElement>
ToLookup<TSource,TKey,TElement> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` IEnumerable<TSource>

The IEnumerable<T> to create a Lookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

Returns

ILookup<TKey,TElement>

A Lookup<TKey,TElement> that contains values of type `TElement` selected from the input sequence.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

Examples

The following code example demonstrates how to use ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>) to create a Lookup<TKey,TElement> by using a key selector function and an element selector function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
    public long TrackingNumber { get; set; }
}

public static void ToLookupEx1()
{
    // Create a list of Packages.
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard",
                  Weight = 25.2, TrackingNumber = 89453312L },
              new Package { Company = "Lucerne Publishing",
                  Weight = 18.7, TrackingNumber = 89112755L },
              new Package { Company = "Wingtip Toys",
                  Weight = 6.0, TrackingNumber = 299456122L },
              new Package { Company = "Contoso Pharmaceuticals",
                  Weight = 9.3, TrackingNumber = 670053128L },
              new Package { Company = "Wide World Importers",
                  Weight = 33.8, TrackingNumber = 4665518773L } };

    // Create a Lookup to organize the packages.
    // Use the first character of Company as the key value.
    // Select Company appended to TrackingNumber
    // as the element values of the Lookup.
    ILookup<char, string> lookup =
        packages
        .ToLookup(p => p.Company[0],
                  p => p.Company + " " + p.TrackingNumber);

    // Iterate through each IGrouping in the Lookup.
    foreach (IGrouping<char, string> packageGroup in lookup)
    {
        // Print the key value of the IGrouping.
        Console.WriteLine(packageGroup.Key);
        // Iterate through each value in the
        // IGrouping and print its value.
        foreach (string str in packageGroup)
            Console.WriteLine("    {0}", str);
    }
}

/*
 This code produces the following output:
```

```
 C
     Coho Vineyard 89453312
     Contoso Pharmaceuticals 670053128
 L
     Lucerne Publishing 89112755
 W
     Wingtip Toys 299456122
     Wide World Importers 4665518773
*/
```

Remarks

The ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>) method returns a Lookup<TKey,TElement>, a one-to-many dictionary that maps keys to collections of values. A Lookup<TKey,TElement> differs from a Dictionary<TKey,TValue>, which performs a one-to-one mapping of keys to single values.

The default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)

Source: Lookup.cs

Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.

C#

```
public static System.Linq.ILookup<TKey,TElement>
ToLookup<TSource,TKey,TElement> (this
System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector, Func<TSource,TElement> elementSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` IEnumerable<TSource>

The IEnumerable<T> to create a Lookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

ILookup<TKey,TElement>

A Lookup<TKey,TElement> that contains values of type `TElement` selected from the input sequence.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

Remarks

The ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>) method returns a Lookup<TKey,TElement>, a one-to-many dictionary that maps keys to collections of values. A Lookup<TKey,TElement> differs from a Dictionary<TKey,TValue>, which performs a one-to-one mapping of keys to single values.

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: Lookup.cs

Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.

```
C#
```

```
public static System.Linq.ILookup<TKey,TSource> ToLookup<TSource,TKey>
(this System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

The IEnumerable<T> to create a Lookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

ILookup<TKey,TSource>

A Lookup<TKey,TElement> that contains keys and values. The values within each group are in the same order as in `source`.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

The ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>) method returns a Lookup<TKey,TElement>, a one-to-many dictionary that maps keys to collections of values. A Lookup<TKey,TElement>

is different to a Dictionary<TKey,TValue>, which performs a one-to-one mapping of keys to single values.

If `comparer` is `null`, the default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

ToLookup<TSource,TKey> (IEnumerable<TSource>, Func<TSource,TKey>)

Source: Lookup.cs

Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.

C#

```
public static System.Linq.ILookup<TKey,TSource> ToLookup<TSource,TKey>
(this System.Collections.Generic.IEnumerable<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` IEnumerable<TSource>

The IEnumerable<T> to create a Lookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

Returns

ILookup<TKey,TSource>

A Lookup<TKey,TElement> that contains keys and values. The values within each group are in the same order as in `source`.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

The ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) method returns a Lookup<TKey,TElement>, a one-to-many dictionary that maps keys to collections of values. A Lookup<TKey,TElement> differs from a Dictionary<TKey,TValue>, which performs a one-to-one mapping of keys to single values.

The default equality comparer Default is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Enumerable.TryGetNonEnumeratedCount<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Count.cs

Attempts to determine the number of elements in a sequence without forcing an enumeration.

C#

```
public static bool TryGetNonEnumeratedCount<TSource> (this
System.Collections.Generic.IEnumerable<TSource> source, out int count);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

A sequence that contains elements to be counted.

`count` Int32

When this method returns, contains the number of elements in `source`, or 0 if the count couldn't be determined without enumeration.

Returns

Boolean

`true` if the count of `source` can be determined without enumeration; otherwise, `false`.

Remarks

The method performs a series of type tests, identifying common subtypes whose count can be determined without enumerating. This includes ICollection<T>, ICollection, and internal types used in the LINQ implementation.

The method is typically a constant-time operation, but ultimately this depends on the complexity characteristics of the underlying collection's implementation.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Enumerable.Union Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Produces the set union of two sequences.

Overloads

Expand table

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)

Source: Union.cs

Produces the set union of two sequences by using the default equality comparer.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Union<TSource> (this System.Collections.Generic.IEnumerable<TSource>
first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the first set for the union.

`second` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the second set for the union.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following code example demonstrates how to use `Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` to obtain the union of two sequences of integers.

C#

```
int[] ints1 = { 5, 3, 9, 7, 5, 9, 3, 7 };
int[] ints2 = { 8, 3, 6, 4, 4, 9, 1, 0 };

IEnumerable<int> union = ints1.Union(ints2);

foreach (int num in union)
{
    Console.Write("{0} ", num);
}

/*
 This code produces the following output:

 5 3 9 7 8 6 4 1 0
*/
```

If you want to compare sequences of objects of some custom data type, you have to implement the IEquatable<T> generic interface in a helper class. The following code

example shows how to implement this interface in a custom data type and override GetHashCode and Equals methods.

C#

```
public class ProductA : IEquatable<ProductA>
{
    public string Name { get; set; }
    public int Code { get; set; }

    public bool Equals(ProductA other)
    {
        if (other is null)
            return false;

        return this.Name == other.Name && this.Code == other.Code;
    }

    public override bool Equals(object obj) => Equals(obj as ProductA);
    public override int GetHashCode() => (Name, Code).GetHashCode();
}
```

After you implement this interface, you can use sequences of `ProductA` objects in the `Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)` method, as shown in the following example:

C#

```
ProductA[] store1 = { new ProductA { Name = "apple", Code = 9 },
                      new ProductA { Name = "orange", Code = 4 } };

ProductA[] store2 = { new ProductA { Name = "apple", Code = 9 },
                      new ProductA { Name = "lemon", Code = 12 } };
```

C#

```
//Get the products from the both arrays
//excluding duplicates.

IEnumerable<ProductA> union =
  store1.Union(store2);

foreach (var product in union)
    Console.WriteLine(product.Name + " " + product.Code);

/*
    This code produces the following output:

    apple 9
    orange 4
```

```
    lemon 12
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

This method excludes duplicates from the return set. This is different behavior to the Concat method, which returns all the elements in the input sequences including duplicates.

The default equality comparer, Default, is used to compare values of the types that implement the IEqualityComparer<T> generic interface. To compare a custom data type, you need to implement this interface and provide your own GetHashCode and Equals methods for the type.

When the object returned by this method is enumerated, `Union` enumerates `first` and `second` in that order and yields each element that has not already been yielded.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Union.cs

Produces the set union of two sequences by using a specified IEqualityComparer<T>.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Union<TSource> (this System.Collections.Generic.IEnumerable<TSource>
first, System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

first IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the first set for the union.

second IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the second set for the union.

comparer IEqualityComparer<TSource>

The IEqualityComparer<T> to compare values.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following example shows how to implement an equality comparer that can be used in the Union method.

C#

```
public class Product
{
    public string Name { get; set; }
    public int Code { get; set; }
}

// Custom comparer for the Product class
class ProductComparer : IEqualityComparer<Product>
{
    // Products are equal if their names and product numbers are equal.
    public bool Equals(Product x, Product y)
    {

        //Check whether the compared objects reference the same data.
        if (Object.ReferenceEquals(x, y)) return true;

        //Check whether any of the compared objects is null.
        if (Object.ReferenceEquals(x, null) || Object.ReferenceEquals(y,
null))
            return false;

        //Check whether the products' properties are equal.
        return x.Code == y.Code && x.Name == y.Name;
    }

    // If Equals() returns true for a pair of objects
    // then GetHashCode() must return the same value for these objects.

    public int GetHashCode(Product product)
    {
        //Check whether the object is null
        if (Object.ReferenceEquals(product, null)) return 0;

        //Get hash code for the Name field if it is not null.
        int hashProductName = product.Name == null ? 0 :
product.Name.GetHashCode();

        //Get hash code for the Code field.
        int hashProductCode = product.Code.GetHashCode();

        //Calculate the hash code for the product.
        return hashProductName ^ hashProductCode;
    }
}
```

After you implement this comparer, you can use sequences of `Product` objects in the Union method, as shown in the following example:

```C#
Product[] store10 = { new Product { Name = "apple", Code = 9 },
                      new Product { Name = "orange", Code = 4 } };

Product[] store20 = { new Product { Name = "apple", Code = 9 },
                      new Product { Name = "lemon", Code = 12 } };

//Get the products from the both arrays
//excluding duplicates.

IEnumerable<Product> union =
  store10.Union(store20, new ProductComparer());

foreach (Product product in union)
    Console.WriteLine(product.Name + " " + product.Code);

/*
    This code produces the following output:

    apple 9
    orange 4
    lemon 12
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

When the object returned by this method is enumerated, Union enumerates `first` and `second` in that order and yields each element that has not already been yielded.

The Concat method differs from the Union method because the Concat method returns all the elements in the input sequences including duplicates, whereas Union returns only unique values.

See also

- UnionBy

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.UnionBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.

UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)

Source: Union.cs

Produces the set union of two sequences according to a specified key selector function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
UnionBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second,
Func<TSource,TKey> keySelector);
```

Type Parameters

TSource

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the first set for the union.

`second` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the second set for the union.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The default equality comparer, Default, is used to compare values.

When the object returned by this method is enumerated, UnionBy enumerates `first` and `second` in that order and yields each element that has not already been yielded.

See also

- Union and UnionBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

UnionBy<TSource,TKey> (IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: Union.cs

Produces the set union of two sequences according to a specified key selector function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
UnionBy<TSource,TKey> (this
System.Collections.Generic.IEnumerable<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`first` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the first set for the union.

`second` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the second set for the union.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

The IEqualityComparer<T> to compare values.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

If `comparer` is `null`, the default equality comparer, Default, is used to compare values.

When the object returned by this method is enumerated, UnionBy enumerates `first` and `second` in that order and yields each element that has not already been yielded.

See also

- Union and UnionBy examples

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Enumerable.Where Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Filters a sequence of values based on a predicate.

Overloads

Expand table

Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)

Source: Where.cs

Filters a sequence of values based on a predicate.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Where<TSource> (this System.Collections.Generic.IEnumerable<TSource>
source, Func<TSource,bool> predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to filter.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains elements from the input sequence that satisfy the condition.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use Where<TSource> (IEnumerable<TSource>, Func<TSource,Boolean>) to filter a sequence.

C#

```
List<string> fruits =
    new List<string> { "apple", "passionfruit", "banana", "mango",
                       "orange", "blueberry", "grape", "strawberry" };

IEnumerable<string> query = fruits.Where(fruit => fruit.Length < 6);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}
/*
 This code produces the following output:

 apple
 mango
 grape
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

In query expression syntax, a `where` (C#) or `Where` (Visual Basic) clause translates to an invocation of Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>).

See also

- where clause (C# Reference)
- Where Clause (Visual Basic)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)

Source: Where.cs

Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
Where<TSource> (this System.Collections.Generic.IEnumerable<TSource>
```

```
source, Func<TSource,int,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to filter.

`predicate` Func<TSource,Int32,Boolean>

A function to test each source element for a condition; the second parameter of the function represents the index of the source element.

Returns

IEnumerable<TSource>

An IEnumerable<T> that contains elements from the input sequence that satisfy the condition.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use Where<TSource> (IEnumerable<TSource>, Func<TSource,Int32,Boolean>) to filter a sequence based on a predicate that involves the index of each element.

C#

```
int[] numbers = { 0, 30, 20, 15, 90, 85, 40, 75 };

IEnumerable<int> query =
    numbers.Where((number, index) => number <= index * 10);

foreach (int number in query)
{
    Console.WriteLine(number);
```

```
}
/*
 This code produces the following output:

 0
 20
 15
 40
*/
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The first argument of `predicate` represents the element to test. The second argument represents the zero-based index of the element within `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Enumerable.Zip Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Overloads

Expand table

Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.

Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)

Source: Zip.cs

Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Zip<TFirst,TSecond,TResult> (this
System.Collections.Generic.IEnumerable<TFirst> first,
System.Collections.Generic.IEnumerable<TSecond> second,
Func<TFirst,TSecond,TResult> resultSelector);
```

Type Parameters

`TFirst`

The type of the elements of the first input sequence.

`TSecond`

The type of the elements of the second input sequence.

`TResult`

The type of the elements of the result sequence.

Parameters

`first` IEnumerable<TFirst>

The first sequence to merge.

`second` IEnumerable<TSecond>

The second sequence to merge.

`resultSelector` Func<TFirst,TSecond,TResult>

A function that specifies how to merge the elements from the two sequences.

Returns

IEnumerable<TResult>

An IEnumerable<T> that contains merged elements of two input sequences.

Exceptions

ArgumentNullException

`first` or `second` is `null`.

Examples

The following code example demonstrates how to use the Zip method to merge two sequences.

```C#
int[] numbers = { 1, 2, 3, 4 };
string[] words = { "one", "two", "three" };
```

```
var numbersAndWords = numbers.Zip(words, (first, second) => first + " " +
second);

foreach (var item in numbersAndWords)
    Console.WriteLine(item);

// This code produces the following output:

// 1 one
// 2 two
// 3 three
```

Remarks

This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using `foreach` in C# or `For Each` in Visual Basic.

The method merges each element of the first sequence with an element that has the same index in the second sequence. If the sequences do not have the same number of elements, the method merges sequences until it reaches the end of one of them. For example, if one sequence has three elements and the other one has four, the result sequence will have only three elements.

See also

- Projection operations - Zip

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Zip<TFirst,TSecond,TThird> (IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)

Source: Zip.cs

Produces a sequence of tuples with elements from the three specified sequences.

C#

```
public static System.Collections.Generic.IEnumerable<(TFirst First,
TSecond Second, TThird Third)> Zip<TFirst,TSecond,TThird> (this
System.Collections.Generic.IEnumerable<TFirst> first,
System.Collections.Generic.IEnumerable<TSecond> second,
System.Collections.Generic.IEnumerable<TThird> third);
```

Type Parameters

`TFirst`

The type of the elements of the first input sequence.

`TSecond`

The type of the elements of the second input sequence.

`TThird`

The type of the elements of the third input sequence.

Parameters

`first` IEnumerable<TFirst>

The first sequence to merge.

`second` IEnumerable<TSecond>

The second sequence to merge.

`third` IEnumerable<TThird>

The third sequence to merge.

Returns

IEnumerable<ValueTuple<TFirst,TSecond,TThird>>

A sequence of tuples with elements taken from the first, second, and third sequences, in that order.

See also

- Projection operations - Zip

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)

Source: Zip.cs

Produces a sequence of tuples with elements from the two specified sequences.

C#

```
public static System.Collections.Generic.IEnumerable<(TFirst First,
TSecond Second)> Zip<TFirst,TSecond> (this
System.Collections.Generic.IEnumerable<TFirst> first,
System.Collections.Generic.IEnumerable<TSecond> second);
```

Type Parameters

`TFirst`

The type of the elements of the first input sequence.

`TSecond`

The type of the elements of the second input sequence.

Parameters

`first` IEnumerable<TFirst>

The first sequence to merge.

`second` IEnumerable<TSecond>

The second sequence to merge.

Returns

IEnumerable<ValueTuple<TFirst,TSecond>>

A sequence of tuples with elements taken from the first and second sequences, in that order.

See also

- Projection operations - Zip

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9

EnumerableExecutor Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableExecutor.cs

Represents an expression tree and provides functionality to execute the expression tree after rewriting it.

C#

```
public abstract class EnumerableExecutor
```

Inheritance Object → EnumerableExecutor

Derived System.Linq.EnumerableExecutor<T>

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableExecutor<T> Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableExecutor.cs

Represents an expression tree and provides functionality to execute the expression tree after rewriting it.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
public class EnumerableExecutor<T> : System.Linq.EnumerableExecutor
```

Type Parameters

T

The data type of the value that results from executing the expression tree.

Inheritance Object → EnumerableExecutor → EnumerableExecutor<T>

Constructors

Expand table

EnumerableExecutor<T>(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Initializes a new instance of the EnumerableExecutor<T> class.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EnumerableExecutor<T>(Expression) Constructor

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableExecutor.cs

Initializes a new instance of the EnumerableExecutor<T> class.

This API supports the product infrastructure and is not intended to be used directly from your code.

```
C#
public EnumerableExecutor (System.Linq.Expressions.Expression expression);
```

Parameters

`expression` Expression

An expression tree to associate with the new instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Represents an IEnumerable as an EnumerableQuery data source.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
public abstract class EnumerableQuery
```

Inheritance Object → EnumerableQuery

Derived System.Linq.EnumerableQuery<T>

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery<T> Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Represents an IEnumerable<T> collection as an IQueryable<T> data source.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
public class EnumerableQuery<T> : System.Linq.EnumerableQuery,
System.Collections.Generic.IEnumerable<T>, System.Linq.IOrderedQueryable<T>,
System.Linq.IQueryable<T>, System.Linq.IQueryProvider
```

Type Parameters

T

The type of the data in the collection.

Inheritance Object → EnumerableQuery → EnumerableQuery<T>

Implements IEnumerable<T> , IEnumerable , IOrderedQueryable , IOrderedQueryable<T> , IQueryable , IQueryable<T> , IQueryProvider

Constructors

Expand table

EnumerableQuery<T> (Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Initializes a new instance of the EnumerableQuery<T> class and associates the instance with an expression tree.

EnumerableQuery<T> (IEnumerable<T>)	This API supports the product infrastructure and is not intended to be used directly from your code. Initializes a new instance of the EnumerableQuery<T> class and associates it with an IEnumerable<T> collection.

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
Memberwise Clone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	This API supports the product infrastructure and is not intended to be used directly from your code. Returns a textual representation of the enumerable collection or, if it is null, of the expression tree that is associated with this instance.

Explicit Interface Implementations

Expand table

IEnumerable.Get Enumerator()	This API supports the product infrastructure and is not intended to be used directly from your code. Returns an enumerator that can iterate through the associated IEnumerable<T> collection, or, if it is null, through the collection that results from rewriting the associated expression tree as a query on an IEnumerable<T> data source and executing it.
IEnumerable<T>.Get Enumerator()	This API supports the product infrastructure and is not intended to be used directly from your code. Returns an enumerator that can iterate through the associated IEnumerable<T> collection, or, if it is null, through the collection

	that results from rewriting the associated expression tree as a query on an IEnumerable<T> data source and executing it.
IQueryable.ElementType	Gets the type of the data in the collection that this instance represents.
IQueryable.Expression	Gets the expression tree that is associated with or that represents this instance.
IQueryable.Provider	Gets the query provider that is associated with this instance.
IQueryProvider.CreateQuery(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Constructs a new EnumerableQuery<T> object and associates it with a specified expression tree that represents an IQueryable collection of data.
IQueryProvider.CreateQuery<TElement>(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Constructs a new EnumerableQuery<T> object and associates it with a specified expression tree that represents an IQueryable<T> collection of data.
IQueryProvider.Execute(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Executes an expression after rewriting it to call Enumerable methods instead of Queryable methods on any enumerable data sources that cannot be queried by Queryable methods.
IQueryProvider.Execute<TElement>(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Executes an expression after rewriting it to call Enumerable methods instead of Queryable methods on any enumerable data sources that cannot be queried by Queryable methods.

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.

ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary

	of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and

	produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is

	used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each

	element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.

DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a

	sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.

FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key

	selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.

GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T>

	is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.

Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence

	and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.

OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an

	IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an

	exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such

	element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on

	each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform

	function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.

ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization

	of a query.
Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether all the elements of a sequence satisfy a condition.
Any<TSource>(IQueryable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` appended at the end.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a

	generic IQueryable<T>.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the average of a sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the average of a sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the average of a sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the average of a sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the average of a sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the average of a sequence of nullable Double values that is obtained by invoking a projection function on each

	element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the average of a sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the average of a sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the average of a sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the average of a sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Cast<TResult>(IQueryable)	Converts the elements of an IQueryable to the specified type.
Chunk<TSource>(IQueryable<TSource>, Int32)	Split the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IQueryable<TSource>, TSource)	Determines whether a sequence contains a

	specified element by using the default equality comparer.
Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IQueryable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the number of elements in the specified sequence that satisfies a condition.
DefaultIfEmpty<TSource>(IQueryable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IQueryable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns distinct elements from a sequence according to

	a specified key selector function.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
ElementAt<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set difference of two

	sequences according to a specified key selector function.
First<TSource>(IQueryable<TSource>)	Returns the first element of a sequence.
First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IQueryable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>)	Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>,	Groups the elements of a sequence according to a specified key

Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>)	selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>)	Correlates the elements of two sequences based on key equality and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set intersection of two sequences according to a specified key selector function.

IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IQueryable<TSource>)	Returns the last element in a sequence.
Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IQueryable<TSource>)	Returns the last element in a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LongCount<TSource>(IQueryable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns an Int64 that represents the number of elements in a sequence that satisfy a condition.
Max<TSource>(IQueryable<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the maximum resulting value.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
Min<TSource>(IQueryable<TSource>)	Returns the minimum value of a generic IQueryable<T>.

Min<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T>.
Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the minimum resulting value.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
OfType<TResult>(IQueryable)	Filters the elements of an IQueryable based on a specified type.
Order<T>(IQueryable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in

	descending order according to a key.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IQueryable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` prepended at the beginning.
Reverse<TSource>(IQueryable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one

	sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> and invokes a result selector function on each element therein. The resulting values from each intermediate sequence are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> that incorporates the index of the source element that produced it. A result selector function is invoked on each element of each intermediate sequence, and the resulting values are combined into a single, one-dimensional sequence and returned.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by using the default equality comparer to compare elements.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by using a specified IEqualityComparer<T> to compare elements.

Single<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the only element of a sequence that satisfies a specified condition or a

	default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IQueryable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the elements from `source` with the last `count` elements of the source queryable sequence omitted.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the sum of the sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the sum of the sequence of Double values that is obtained by invoking a

	projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the sum of the sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the sum of the sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the sum of the sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the sum of the sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the sum of the sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the sum of the sequence of nullable Int64 values that is obtained by

	invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the sum of the sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the sum of the sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Take<TSource>(IQueryable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IQueryable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the last `count` elements from `source`.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.
ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Filters a sequence of values based on a

	predicate.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IQueryable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IQueryable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IQueryable<TFirst>, IEnumerable<TSecond>, Expression<Func<TFirst,TSecond,TResult>>)	Merges two sequences by using the specified predicate function.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document

	in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

EnumerableQuery<T> Constructors

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Initializes a new instance of the EnumerableQuery<T> class.

Overloads

Expand table

EnumerableQuery<T>(IEnumerable<T>)	This API supports the product infrastructure and is not intended to be used directly from your code. Initializes a new instance of the EnumerableQuery<T> class and associates it with an IEnumerable<T> collection.
EnumerableQuery<T>(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Initializes a new instance of the EnumerableQuery<T> class and associates the instance with an expression tree.

EnumerableQuery<T>(IEnumerable<T>)

Source: EnumerableQuery.cs

Initializes a new instance of the EnumerableQuery<T> class and associates it with an IEnumerable<T> collection.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
public EnumerableQuery (System.Collections.Generic.IEnumerable<T>
enumerable);
```

Parameters

`enumerable` IEnumerable<T>

A collection to associate with the new instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

EnumerableQuery<T>(Expression)

Source: EnumerableQuery.cs

Initializes a new instance of the EnumerableQuery<T> class and associates the instance with an expression tree.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
public EnumerableQuery (System.Linq.Expressions.Expression expression);
```

Parameters

`expression` Expression

An expression tree to associate with the new instance.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery<T>.ToString Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Returns a textual representation of the enumerable collection or, if it is null, of the expression tree that is associated with this instance.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
public override string? ToString ();
```

Returns

String

A textual representation of the enumerable collection or, if it is null, of the expression tree that is associated with this instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on

.NET feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery<T>.IEnumerable<T>.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Returns an enumerator that can iterate through the associated IEnumerable<T> collection, or, if it is null, through the collection that results from rewriting the associated expression tree as a query on an IEnumerable<T> data source and executing it.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
System.Collections.Generic.IEnumerator<T> IEnumerable<T>.GetEnumerator ();
```

Returns

IEnumerator<T>

An enumerator that can be used to iterate through the associated data source.

Implements

GetEnumerator()

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EnumerableQuery<T>.IEnumerable.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Returns an enumerator that can iterate through the associated IEnumerable<T> collection, or, if it is null, through the collection that results from rewriting the associated expression tree as a query on an IEnumerable<T> data source and executing it.

This API supports the product infrastructure and is not intended to be used directly from your code.

```
C#

System.Collections.IEnumerator IEnumerable.GetEnumerator ();
```

Returns

IEnumerator

An enumerator that can be used to iterate through the associated data source.

Implements

GetEnumerator()

Remarks

This member is an explicit interface member implementation. It can be used only when the EnumerableQuery<T> instance is cast to an IEnumerable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery<T>.IQueryable.ElementType Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Gets the type of the data in the collection that this instance represents.

```
C#

Type System.Linq.IQueryable.ElementType { get; }
```

Property Value

Type

The type of the data in the collection that this instance represents.

Implements

ElementType

Remarks

This member is an explicit interface member implementation. It can be used only when the EnumerableQuery<T> instance is cast to an IQueryable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery<T>.IQueryable.Expression Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Gets the expression tree that is associated with or that represents this instance.

C#

```
System.Linq.Expressions.Expression System.Linq.IQueryable.Expression { get;
}
```

Property Value

Expression

The expression tree that is associated with or that represents this instance.

Implements

Expression

Remarks

This member is an explicit interface member implementation. It can be used only when the EnumerableQuery<T> instance is cast to an IQueryable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

EnumerableQuery<T>.IQueryable.Provider Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: EnumerableQuery.cs

Gets the query provider that is associated with this instance.

```
C#
System.Linq.IQueryProvider System.Linq.IQueryable.Provider { get; }
```

Property Value

IQueryProvider

The query provider that is associated with this instance.

Implements

Provider

Remarks

This member is an explicit interface member implementation. It can be used only when the EnumerableQuery<T> instance is cast to an IQueryable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

EnumerableQuery<T>.IQueryProvider.CreateQuery Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

IQueryProvider.CreateQuery(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Constructs a new EnumerableQuery<T> object and associates it with a specified expression tree that represents an IQueryable collection of data.
IQueryProvider.CreateQuery<TElement>(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Constructs a new EnumerableQuery<T> object and associates it with a specified expression tree that represents an IQueryable<T> collection of data.

IQueryProvider.CreateQuery(Expression)

Source: EnumerableQuery.cs

Constructs a new EnumerableQuery<T> object and associates it with a specified expression tree that represents an IQueryable collection of data.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
System.Linq.IQueryable IQueryProvider.CreateQuery
(System.Linq.Expressions.Expression expression);
```

Parameters

`expression` Expression

An expression tree that represents an IQueryable collection of data.

Returns

IQueryable

An EnumerableQuery<T> object that is associated with `expression`.

Implements

CreateQuery(Expression)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

IQueryProvider.CreateQuery<TElement>(Expression)

Source: EnumerableQuery.cs

Constructs a new EnumerableQuery<T> object and associates it with a specified expression tree that represents an IQueryable<T> collection of data.

This API supports the product infrastructure and is not intended to be used directly from your code.

Type Parameters

`S`

The type of the data in the collection that `expression` represents.

`TElement`

The type of the elements of the IQueryable<T> that is returned.

Parameters

`expression` Expression

An expression tree to execute.

Returns

IQueryable<TElement>

An EnumerableQuery object that is associated with `expression`.

Implements

CreateQuery<TElement>(Expression)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

EnumerableQuery<T>.IQueryProvider.Execute Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

IQueryProvider.Execute(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Executes an expression after rewriting it to call Enumerable methods instead of Queryable methods on any enumerable data sources that cannot be queried by Queryable methods.
IQueryProvider.Execute<TElement>(Expression)	This API supports the product infrastructure and is not intended to be used directly from your code. Executes an expression after rewriting it to call Enumerable methods instead of Queryable methods on any enumerable data sources that cannot be queried by Queryable methods.

IQueryProvider.Execute(Expression)

Source: EnumerableQuery.cs

Executes an expression after rewriting it to call Enumerable methods instead of Queryable methods on any enumerable data sources that cannot be queried by Queryable methods.

This API supports the product infrastructure and is not intended to be used directly from your code.

C#

```
object IQueryProvider.Execute (System.Linq.Expressions.Expression
expression);
```

Parameters

`expression` Expression

An expression tree to execute.

Returns

Object

The value that results from executing `expression`.

Implements

Execute(Expression)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

IQueryProvider.Execute<TElement>(Expression)

Source: EnumerableQuery.cs

Executes an expression after rewriting it to call Enumerable methods instead of Queryable methods on any enumerable data sources that cannot be queried by Queryable methods.

This API supports the product infrastructure and is not intended to be used directly from your code.

Type Parameters

`S`

The type of the data in the collection that `expression` represents.

`TElement`

Parameters

`expression` Expression

An expression tree to execute.

Returns

TElement

The value that results from executing `expression`.

Implements

Execute<TResult>(Expression)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IGrouping<TKey,TElement> Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Represents a collection of objects that have a common key.

C#

```
public interface IGrouping<out TKey,out TElement> :
System.Collections.Generic.IEnumerable<out TElement>
```

Type Parameters

TKey

The type of the key of the IGrouping<TKey,TElement>.
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.

TElement

The type of the values in the IGrouping<TKey,TElement>.
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.

Implements IEnumerable<TElement> , IEnumerable

Examples

The following example demonstrates how to work with an IGrouping<TKey,TElement> object.

In this example, GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) is called on the array of MemberInfo objects returned by GetMembers. GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) groups the

objects based on the value of their MemberType property. Each unique value for MemberType in the array of MemberInfo objects becomes a key for a new IGrouping<TKey,TElement> object, and the MemberInfo objects that have that key form the IGrouping<TKey,TElement> object's sequence of values.

Finally, the First method is called on the sequence of IGrouping<TKey,TElement> objects to obtain just the first IGrouping<TKey,TElement> object.

The example then outputs the key of the IGrouping<TKey,TElement> object and the Name property of each value in the IGrouping<TKey,TElement> object's sequence of values. Notice that to access an IGrouping<TKey,TElement> object's sequence of values, you simply use the IGrouping<TKey,TElement> variable itself.

C#

```
// Get an IGrouping object.
IGrouping<System.Reflection.MemberTypes, System.Reflection.MemberInfo> group
=
    typeof(String).GetMembers().
    GroupBy(member => member.MemberType).
    First();

// Output the key of the IGrouping, then iterate
// through each value in the sequence of values
// of the IGrouping and output its Name property.
Console.WriteLine("\nValues that have the key '{0}':", group.Key);
foreach (System.Reflection.MemberInfo mi in group)
    Console.WriteLine(mi.Name);

// The output is similar to:

// Values that have the key 'Method':
// get_Chars
// get_Length
// IndexOf
// IndexOfAny
// LastIndexOf
// LastIndexOfAny
// Insert
// Replace
// Replace
// Remove
// Join
// Join
// Equals
// Equals
// Equals
// ...
```

Remarks

An IGrouping<TKey,TElement> is an IEnumerable<T> that additionally has a key. The key represents the attribute that is common to each value in the IGrouping<TKey,TElement>.

The values of an IGrouping<TKey,TElement> are accessed much as the elements of an IEnumerable<T> are accessed. For example, you can access the values by using a `foreach` in Visual C# or `For Each` in Visual Basic loop to iterate through the IGrouping<TKey,TElement> object. The Example section contains a code example that shows you how to access both the key and the values of an IGrouping<TKey,TElement> object.

The IGrouping<TKey,TElement> type is used by the GroupBy standard query operator methods, which return a sequence of elements of type IGrouping<TKey,TElement>.

Properties

Expand table

Key	Gets the key of the IGrouping<TKey,TElement>.

Methods

Expand table

GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>,	Creates a FrozenDictionary<TKey,TValu

Func<TSource,TElement>, IEqualityComparer<TKey>)	e> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEquality Comparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of

	its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.

Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.

Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.

DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.

First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using

	the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a

	condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.

Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a

	sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending

	order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.

SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a

	specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each

	element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each

	element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.

ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.

AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.

InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

IGrouping<TKey,TElement>.Key Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Gets the key of the IGrouping<TKey,TElement>.

C#

```
public TKey Key { get; }
```

Property Value

TKey

The key of the IGrouping<TKey,TElement>.

Examples

The following example demonstrates how to use the Key property to label each IGrouping<TKey,TElement> object in a sequence of IGrouping<TKey,TElement> objects. The GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>) method is used to obtain a sequence of IGrouping<TKey,TElement> objects. The `foreach` in Visual C# or `For Each` in Visual Basic loop then iterates through each IGrouping<TKey,TElement> object, outputting its key and the number of values it contains.

C#

```
// Get a sequence of IGrouping objects.
IEnumerable<IGrouping<System.Reflection.MemberTypes,
System.Reflection.MemberInfo>> memberQuery =
    typeof(String).GetMembers().
    GroupBy(member => member.MemberType);

// Output the key of each IGrouping object and the count of values.
foreach (IGrouping<System.Reflection.MemberTypes,
```

```
System.Reflection.MemberInfo> group in memberQuery)
    Console.WriteLine("(Key) {0} (Member count) {1}", group.Key,
group.Count());

// The output is similar to:
// (Key) Method (Member count) 113
// (Key) Constructor (Member count) 8
// (Key) Property (Member count) 2
// (Key) Field (Member count) 1
```

Remarks

The key of an IGrouping<TKey,TElement> represents the attribute that is common to each value in the IGrouping<TKey,TElement>.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ILookup<TKey,TElement> Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Defines an indexer, size property, and Boolean search method for data structures that map keys to IEnumerable<T> sequences of values.

C#

```
public interface ILookup<TKey,TElement> :
System.Collections.Generic.IEnumerable<System.Linq.IGrouping<TKey,TElement>>
```

Type Parameters

TKey

The type of the keys in the ILookup<TKey,TElement>.

TElement

The type of the elements in the IEnumerable<T> sequences that make up the values in the ILookup<TKey,TElement>.

Derived System.Linq.Lookup<TKey,TElement>

Implements IEnumerable<IGrouping<TKey,TElement>> , IEnumerable

Examples

The following code example creates an ILookup<TKey,TElement> object and iterates through its contents.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
    public long TrackingNumber { get; set; }
```

```
}

public static void ILookupExample()
{
    // Create a list of Packages to put into an ILookup data structure.
    List<Package> packages = new List<Package> { new Package { Company =
"Coho Vineyard", Weight = 25.2, TrackingNumber = 89453312L },
                                                 new Package { Company =
"Lucerne Publishing", Weight = 18.7, TrackingNumber = 89112755L },
                                                 new Package { Company =
"Wingtip Toys", Weight = 6.0, TrackingNumber = 299456122L },
                                                 new Package { Company =
"Contoso Pharmaceuticals", Weight = 9.3, TrackingNumber = 670053128L },
                                                 new Package { Company =
"Wide World Importers", Weight = 33.8, TrackingNumber = 4665518773L } };

    // Create a Lookup to organize the packages. Use the first character of
Company as the key value.
    // Select Company appended to TrackingNumber for each element value in
the ILookup object.
    ILookup<char, string> packageLookup = packages.ToLookup(
        p => p.Company[0],
        p => p.Company + " " + p.TrackingNumber
        );

    // Iterate through each value in the ILookup and output the contents.
    foreach (var packageGroup in packageLookup)
    {
        // Print the key value.
        Console.WriteLine(packageGroup.Key);
        // Iterate through each value in the collection.
        foreach (string str in packageGroup)
            Console.WriteLine("    {0}", str);
    }

    // This code produces the following output:
    //
    // C
    //     Coho Vineyard 89453312
    //     Contoso Pharmaceuticals 670053128
    // L
    //     Lucerne Publishing 89112755
    // W
    //     Wingtip Toys 299456122
    //     Wide World Importers 4665518773
}
```

Remarks

The type Lookup<TKey,TElement> implements the ILookup<TKey,TElement> interface.

The extension method ToLookup, which can be appended to the end of a LINQ query, returns an object of type ILookup<TKey,TElement>.

Properties

Expand table

Count	Gets the number of key/value collection pairs in the ILookup<TKey,TElement>.
Item[TKey]	Gets the IEnumerable<T> sequence of values indexed by a specified key.

Methods

Expand table

Contains(TKey)	Determines whether a specified key exists in the ILookup<TKey,TElement>.
GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.

ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary

	of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value,

	and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal

	values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.

Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEquality Comparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEquality Comparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector

	function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.

FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>,	Groups the elements of a sequence according to a specified key selector function

Func<TKey,IEnumerable<TSource>,TResult>, IEquality Comparer<TKey>)	and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.

LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the

	maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according

	to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.

Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by

	using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence

	and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.

TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.

ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.

UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.

Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.

Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Lookup<TKey,TElement>

ILookup<TKey,TElement>.Count Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Gets the number of key/value collection pairs in the ILookup<TKey,TElement>.

C#

```
public int Count { get; }
```

Property Value

Int32

The number of key/value collection pairs in the ILookup<TKey,TElement>.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ILookup<TKey,TElement>.Item[TKey] Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Gets the IEnumerable<T> sequence of values indexed by a specified key.

C#

```
public System.Collections.Generic.IEnumerable<TElement> this[TKey key] {
get; }
```

Parameters

key TKey

The key of the desired sequence of values.

Property Value

IEnumerable<TElement>

The IEnumerable<T> sequence of values indexed by the specified key.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ILookup<TKey,TElement>.Contains(TKey) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Determines whether a specified key exists in the ILookup<TKey,TElement>.

C#

```
public bool Contains (TKey key);
```

Parameters

key TKey

The key to search for in the ILookup<TKey,TElement>.

Returns

Boolean

`true` if `key` is in the ILookup<TKey,TElement>; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

LINQ extension method overrides that offer greater efficiency for ImmutableArray<T> than the standard LINQ methods

NuGet package: System.Collections.Immutable (about immutable collections and how to install)

C#

```
public static class ImmutableArrayExtensions
```

Inheritance Object → ImmutableArrayExtensions

Methods

Expand table

Aggregate<T>(ImmutableArray<T>, Func<T,T,T>)	Applies a function to a sequence of elements in a cumulative way.
Aggregate<TAccumulate,T>(ImmutableArray<T>, TAccumulate, Func<TAccumulate,T,TAccumulate>)	Applies a function to a sequence of elements in a cumulative way.
Aggregate<TAccumulate,TResult,T>(ImmutableArray<T>, TAccumulate, Func<TAccumulate,T,TAccumulate>, Func<TAccumulate,TResult>)	Applies a function to a sequence of elements in a cumulative way.
All<T>(ImmutableArray<T>, Func<T,Boolean>)	Gets a value indicating whether all elements in this array match a given condition.
Any<T>(ImmutableArray<T>)	Gets a value indicating whether the array contains any elements.

Any<T>(ImmutableArray<T>, Func<T,Boolean>)	Gets a value indicating whether the array contains any elements that match a specified condition.
Any<T>(ImmutableArray<T>.Builder)	Returns a value indicating whether this collection contains any elements.
ElementAt<T>(ImmutableArray<T>, Int32)	Returns the element at a specified index in the array.
ElementAtOrDefault<T>(ImmutableArray<T>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
First<T>(ImmutableArray<T>)	Returns the first element in an array.
First<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
First<T>(ImmutableArray<T>.Builder)	Returns the first element in the collection.
FirstOrDefault<T>(ImmutableArray<T>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<T>(ImmutableArray<T>.Builder)	Returns the first element in the collection, or the default value if the collection is empty.
Last<T>(ImmutableArray<T>)	Returns the last element of the array.
Last<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
Last<T>(ImmutableArray<T>.Builder)	Returns the last element in the collection.
LastOrDefault<T>(ImmutableArray<T>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.

LastOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<T>(ImmutableArray<T>.Builder)	Returns the last element in the collection, or the default value if the collection is empty.
Select<T,TResult>(ImmutableArray<T>, Func<T,TResult>)	Projects each element of a sequence into a new form.
SelectMany<TSource,TCollection,TResult>(ImmutableArray<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, IEnumerable<TDerived>, IEqualityComparer<TBase>)	Determines whether two sequences are equal according to an equality comparer.
SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, ImmutableArray<TDerived>, Func<TBase,TBase,Boolean>)	Determines whether two sequences are equal according to an equality comparer.
SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, ImmutableArray<TDerived>, IEqualityComparer<TBase>)	Determines whether two sequences are equal according to an equality comparer.
Single<T>(ImmutableArray<T>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<T>(ImmutableArray<T>)	Returns the only element of the array, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the only element of a sequence that satisfies a specified

	condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
ToArray<T>(ImmutableArray<T>)	Copies the contents of this array to a mutable array.
ToDictionary<TKey,T>(ImmutableArray<T>, Func<T,TKey>)	Creates a dictionary based on the contents of this array.
ToDictionary<TKey,T>(ImmutableArray<T>, Func<T,TKey>, IEqualityComparer<TKey>)	Creates a dictionary based on the contents of this array.
ToDictionary<TKey,TElement,T>(ImmutableArray<T>, Func<T,TKey>, Func<T,TElement>)	Creates a dictionary based on the contents of this array.
ToDictionary<TKey,TElement,T>(ImmutableArray<T>, Func<T,TKey>, Func<T,TElement>, IEquality Comparer<TKey>)	Creates a dictionary based on the contents of this array.
Where<T>(ImmutableArray<T>, Func<T,Boolean>)	Filters a sequence of values based on a predicate.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.Aggregate Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

Aggregate<T>(ImmutableArray<T>, Func<T,T,T>)	Applies a function to a sequence of elements in a cumulative way.
Aggregate<TAccumulate,T>(ImmutableArray<T>, TAccumulate, Func<TAccumulate,T,TAccumulate>)	Applies a function to a sequence of elements in a cumulative way.
Aggregate<TAccumulate,TResult,T>(ImmutableArray<T>, TAccumulate, Func<TAccumulate,T,TAccumulate>, Func<TAccumulate,TResult>)	Applies a function to a sequence of elements in a cumulative way.

Aggregate<T>(ImmutableArray<T>, Func<T,T,T>)

Source: ImmutableArrayExtensions.cs

Applies a function to a sequence of elements in a cumulative way.

C#

```
public static T? Aggregate<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,T,T> func);
```

Type Parameters

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The collection to apply the function to.

`func` Func<T,T,T>

A function to be invoked on each element, in a cumulative way.

Returns

T

The final value after the cumulative function has been applied to all elements.

Remarks

Aggregate method makes it simple to perform a calculation over a sequence of values. This method works by calling `func` one time for each element in source. Each time `func` is called, Aggregate passes both the element from the sequence and an aggregated value (as the first argument to func). The value of the `seed` parameter is used as the initial aggregate value. The result of `func` replaces the previous aggregated value. Aggregate returns the final result of `func`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Aggregate<TAccumulate,T> (ImmutableArray<T>, TAccumulate, Func<TAccumulate,T,TAccumulate>)

Source: ImmutableArrayExtensions.cs

Applies a function to a sequence of elements in a cumulative way.

C#

```
public static TAccumulate Aggregate<TAccumulate,T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
TAccumulate seed, Func<TAccumulate,T,TAccumulate> func);
```

Type Parameters

TAccumulate

The type of the accumulated value.

T

The type of element contained by the collection.

Parameters

immutableArray ImmutableArray<T>

The collection to apply the function to.

seed TAccumulate

The initial accumulator value.

func Func<TAccumulate,T,TAccumulate>

A function to be invoked on each element, in a cumulative way.

Returns

TAccumulate

The final accumulator value.

Remarks

Aggregate method makes it simple to perform a calculation over a sequence of values. This method works by calling `func` one time for each element in source. Each time `func` is called, Aggregate passes both the element from the sequence and an aggregated value (as the first argument to func). The value of the `seed` parameter is used as the initial aggregate value. The result of `func` replaces the previous aggregated value. Aggregate returns the final result of `func`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Aggregate<TAccumulate,TResult,T> (ImmutableArray<T>, TAccumulate, Func<TAccumulate,T,TAccumulate>, Func<TAccumulate,TResult>)

Source: ImmutableArrayExtensions.cs

Applies a function to a sequence of elements in a cumulative way.

C#

```
public static TResult Aggregate<TAccumulate,TResult,T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
TAccumulate seed, Func<TAccumulate,T,TAccumulate> func,
Func<TAccumulate,TResult> resultSelector);
```

Type Parameters

`TAccumulate`

The type of the accumulated value.

`TResult`

The type of result returned by the result selector.

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The collection to apply the function to.

`seed` TAccumulate

The initial accumulator value.

`func` Func<TAccumulate,T,TAccumulate>

A function to be invoked on each element, in a cumulative way.

`resultSelector` Func<TAccumulate,TResult>

A function to transform the final accumulator value into the result type.

Returns

TResult

The final accumulator value.

Remarks

Aggregate method makes it simple to perform a calculation over a sequence of values. This method works by calling `func` one time for each element in source. Each time `func` is called, Aggregate passes both the element from the sequence and an aggregated value (as the first argument to func). The value of the `seed` parameter is used as the initial aggregate value. The result of `func` replaces the previous aggregated value. Aggregate returns the final result of `func`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.All<T> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Gets a value indicating whether all elements in this array match a given condition.

C#

```
public static bool All<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray, Func<T,bool>
predicate);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

immutableArray ImmutableArray<T>

The array to check for matches.

predicate Func<T,Boolean>

The predicate.

Returns

Boolean

`true` if every element of the source sequence passes the test in the specified predicate; otherwise, `false`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.Any Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

Any<T>(ImmutableArray<T>)	Gets a value indicating whether the array contains any elements.
Any<T>(ImmutableArray<T>.Builder)	Returns a value indicating whether this collection contains any elements.
Any<T>(ImmutableArray<T>, Func<T,Boolean>)	Gets a value indicating whether the array contains any elements that match a specified condition.

Any<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Gets a value indicating whether the array contains any elements.

C#

```
public static bool Any<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to check for elements.

Returns

Boolean

`true` if the array contains an elements; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Any<T>(ImmutableArray<T>.Builder)

Source: ImmutableArrayExtensions.cs

Returns a value indicating whether this collection contains any elements.

C#

```
public static bool Any<T> (this
System.Collections.Immutable.ImmutableArray<T>.Builder builder);
```

Type Parameters

`T`

The type of elements in the array.

Parameters

`builder` ImmutableArray<T>.Builder

The builder to check for matches.

Returns

Boolean

`true` if the array builder contains any elements; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Any<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Gets a value indicating whether the array contains any elements that match a specified condition.

C#

```
public static bool Any<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,bool> predicate);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to check for elements.

`predicate` Func<T,Boolean>

The delegate that defines the condition to match to an element.

Returns

Boolean

`true` if an element matches the specified condition; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.ElementAt<T> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Returns the element at a specified index in the array.

C#

```
public static T ElementAt<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray, int index);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

immutableArray ImmutableArray<T>

The array to find an element in.

index Int32

The index for the element to retrieve.

Returns

T

The item at the specified index.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.ElementAtOrDefault<T> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Returns the element at a specified index in a sequence or a default value if the index is out of range.

C#

```
public static T? ElementAtOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray, int index);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

immutableArray ImmutableArray<T>

The array to find an element in.

index Int32

The index for the element to retrieve.

Returns

T

The item at the specified index, or the default value if the index is not found.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.First Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

First<T>(ImmutableArray<T>)	Returns the first element in an array.
First<T>(ImmutableArray<T>.Builder)	Returns the first element in the collection.
First<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.

First<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Returns the first element in an array.

C#

```
public static T First<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

immutableArray ImmutableArray<T>

The array to get an item from.

Returns

T

The first item in the array.

Exceptions

InvalidOperationException

If the array is empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

First<T>(ImmutableArray<T>.Builder)

Source: ImmutableArrayExtensions.cs

Returns the first element in the collection.

C#

```
public static T First<T> (this
System.Collections.Immutable.ImmutableArray<T>.Builder builder);
```

Type Parameters

T

The type of items in the array.

Parameters

`builder` ImmutableArray<T>.Builder

The builder to retrieve an item from.

Returns

T

The first item in the list.

Exceptions

InvalidOperationException

If the array is empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

First<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Returns the first element in a sequence that satisfies a specified condition.

C#

```
public static T First<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,bool> predicate);
```

Type Parameters

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to get an item from.

`predicate` Func<T,Boolean>

The delegate that defines the conditions of the element to search for.

Returns

T

The first item in the list if it meets the condition specified by `predicate`.

Exceptions

InvalidOperationException

If the array is empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.FirstOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

FirstOrDefault<T>(ImmutableArray<T>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<T>(ImmutableArray<T>.Builder)	Returns the first element in the collection, or the default value if the collection is empty.
FirstOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.

FirstOrDefault<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Returns the first element of a sequence, or a default value if the sequence contains no elements.

C#

```
public static T? FirstOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve items from.

Returns

T

The first item in the list, if found; otherwise the default value for the item type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

FirstOrDefault<T> (ImmutableArray<T>.Builder)

Source: ImmutableArrayExtensions.cs

Returns the first element in the collection, or the default value if the collection is empty.

C#

```
public static T? FirstOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T>.Builder builder);
```

Type Parameters

T

The type of item in the builder.

Parameters

`builder` ImmutableArray<T>.Builder

The builder to retrieve an element from.

Returns

T

The first item in the list, if found; otherwise the default value for the item type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

FirstOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.

C#

```
public static T? FirstOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,bool> predicate);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve elements from.

`predicate` Func<T,Boolean>

The delegate that defines the conditions of the element to search for.

Returns

T

The first item in the list, if found; otherwise the default value for the item type.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.Last Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

Last<T>(ImmutableArray<T>.Builder)	Returns the last element in the collection.
Last<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
Last<T>(ImmutableArray<T>)	Returns the last element of the array.

Last<T>(ImmutableArray<T>.Builder)

Source: ImmutableArrayExtensions.cs

Returns the last element in the collection.

C#

```
public static T Last<T> (this
System.Collections.Immutable.ImmutableArray<T>.Builder builder);
```

Type Parameters

T

The type of item in the builder.

Parameters

builder ImmutableArray<T>.Builder

The builder to retrieve elements from.

Returns

T

The last element in the builder.

Exceptions

InvalidOperationException

The collection is empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Last<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Returns the last element of a sequence that satisfies a specified condition.

C#

```
public static T Last<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,bool> predicate);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve elements from.

`predicate` Func<T,Boolean>

The delegate that defines the conditions of the element to retrieve.

Returns

T

The last element of the array that satisfies the `predicate` condition.

Exceptions

InvalidOperationException

The collection is empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Last<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Returns the last element of the array.

```
C#
public static T Last<T> (this
```

```
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

`T`

The type of element contained by the array.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve items from.

Returns

T

The last element in the array.

Exceptions

InvalidOperationException

The collection is empty.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.LastOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

LastOrDefault<T>(ImmutableArray<T>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<T>(ImmutableArray<T>.Builder)	Returns the last element in the collection, or the default value if the collection is empty.
LastOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

LastOrDefault<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Returns the last element of a sequence, or a default value if the sequence contains no elements.

C#

```
public static T? LastOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve items from.

Returns

T

The last element of a sequence, or a default value if the sequence contains no elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

LastOrDefault<T>(ImmutableArray<T>.Builder)

Source: ImmutableArrayExtensions.cs

Returns the last element in the collection, or the default value if the collection is empty.

C#

```
public static T? LastOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T>.Builder builder);
```

Type Parameters

T

The type of item in the builder.

Parameters

`builder` ImmutableArray<T>.Builder

The builder to retrieve an element from.

Returns

T

The last element of a sequence, or a default value if the sequence contains no elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

LastOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

C#

```
public static T? LastOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,bool> predicate);
```

Type Parameters

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve an element from.

`predicate` Func<T,Boolean>

The delegate that defines the conditions of the element to search for.

Returns

T

The last element of a sequence, or a default value if the sequence contains no elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ImmutableArrayExtensions.Select<T,TResult> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Projects each element of a sequence into a new form.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
Select<T,TResult> (this System.Collections.Immutable.ImmutableArray<T>
immutableArray, Func<T,TResult> selector);
```

Type Parameters

`T`

The type of element contained by the collection.

`TResult`

The type of the result element.

Parameters

`immutableArray` ImmutableArray<T>

The immutable array to select items from.

`selector` Func<T,TResult>

A transform function to apply to each element.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the transform function on each element of source.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.SelectMany<TSource,TCollection,TResult> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.

C#

```
public static System.Collections.Generic.IEnumerable<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Collections.Immutable.ImmutableArray<TSource> immutableArray,
Func<TSource,System.Collections.Generic.IEnumerable<TCollection>>
collectionSelector, Func<TSource,TCollection,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `immutableArray`.

`TCollection`

The type of the intermediate elements collected by `collectionSelector`.

`TResult`

The type of the elements of the resulting sequence.

Parameters

`immutableArray` ImmutableArray<TSource>

The immutable array.

`collectionSelector` Func<TSource,IEnumerable<TCollection>>

A transform function to apply to each element of the input sequence.

`resultSelector` Func<TSource,TCollection,TResult>

A transform function to apply to each element of the intermediate sequence.

Returns

IEnumerable<TResult>

An IEnumerable<T> whose elements are the result of invoking the one-to-many transform function `collectionSelector` on each element of `immutableArray` and then mapping each of those sequence elements and their corresponding source element to a result element.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.SequenceEqual Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, IEnumerable<TDerived>, IEqualityComparer<TBase>)	Determines whether two sequences are equal according to an equality comparer.
SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, ImmutableArray<TDerived>, IEqualityComparer<TBase>)	Determines whether two sequences are equal according to an equality comparer.
SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, ImmutableArray<TDerived>, Func<TBase,TBase,Boolean>)	Determines whether two sequences are equal according to an equality comparer.

SequenceEqual<TDerived,TBase>(ImmutableArray<TBase>, IEnumerable<TDerived>, IEqualityComparer<TBase>)

Source: ImmutableArrayExtensions.cs

Determines whether two sequences are equal according to an equality comparer.

C#

```
public static bool SequenceEqual<TDerived,TBase> (this
System.Collections.Immutable.ImmutableArray<TBase> immutableArray,
```

```
System.Collections.Generic.IEnumerable<TDerived> items,
System.Collections.Generic.IEqualityComparer<TBase>? comparer = default)
where TDerived : TBase;
```

Type Parameters

`TDerived`

The type of element in the compared array.

`TBase`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<TBase>

The array to use for comparison.

`items` IEnumerable<TDerived>

The items to use for comparison.

`comparer` IEqualityComparer<TBase>

The comparer to use to check for equality.

Returns

Boolean

`true` to indicate the sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0

Product	Versions
UWP	10.0

SequenceEqual<TDerived,TBase> (ImmutableArray<TBase>, ImmutableArray<TDerived>, IEqualityComparer<TBase>)

Source: ImmutableArrayExtensions.cs

Determines whether two sequences are equal according to an equality comparer.

C#

```
public static bool SequenceEqual<TDerived,TBase> (this
System.Collections.Immutable.ImmutableArray<TBase> immutableArray,
System.Collections.Immutable.ImmutableArray<TDerived> items,
System.Collections.Generic.IEqualityComparer<TBase>? comparer = default)
where TDerived : TBase;
```

Type Parameters

`TDerived`

The type of element in the compared array.

`TBase`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<TBase>

The array to use for comparison.

`items` ImmutableArray<TDerived>

The items to use for comparison.

`comparer` IEqualityComparer<TBase>

The comparer to use to check for equality.

Returns

Boolean

`true` to indicate the sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

SequenceEqual<TDerived,TBase> (ImmutableArray<TBase>, ImmutableArray<TDerived>, Func<TBase,TBase,Boolean>)

Source: ImmutableArrayExtensions.cs

Determines whether two sequences are equal according to an equality comparer.

C#

```
public static bool SequenceEqual<TDerived,TBase> (this
System.Collections.Immutable.ImmutableArray<TBase> immutableArray,
System.Collections.Immutable.ImmutableArray<TDerived> items,
Func<TBase,TBase,bool> predicate) where TDerived : TBase;
```

Type Parameters

`TDerived`

The type of element in the compared array.

`TBase`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<TBase>

The array to use for comparison.

`items` ImmutableArray<TDerived>

The items to use for comparison.

`predicate` Func<TBase,TBase,Boolean>

The comparer to use to check for equality.

Returns

Boolean

`true` to indicate the sequences are equal; otherwise, `false`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ImmutableArrayExtensions.Single Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

Single<T>(ImmutableArray<T>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<T>(ImmutableArray<T>, Func<T,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

Single<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.

C#

```
public static T Single<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to retrieve the element from.

Returns

T

The element in the sequence.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Single<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

C#

```
public static T Single<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,bool> predicate);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The immutable array to return a single element from.

`predicate` Func<T,Boolean>

The function to test whether an element should be returned.

Returns

T

Returns Boolean.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.SingleOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

SingleOrDefault<T> (ImmutableArray<T>)	Returns the only element of the array, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<T> (ImmutableArray<T>, Func<T,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

SingleOrDefault<T>(ImmutableArray<T>)

Source: ImmutableArrayExtensions.cs

Returns the only element of the array, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

C#

```
public static T? SingleOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array.

Returns

T

The element in the array, or the default value if the array is empty.

Exceptions

InvalidOperationException

`source` contains more than one element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

SingleOrDefault<T>(ImmutableArray<T>, Func<T,Boolean>)

Source: ImmutableArrayExtensions.cs

Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

C#

```
public static T? SingleOrDefault<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
```

```
Func<T,bool> predicate);
```

Type Parameters

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to get the element from.

`predicate` Func<T,Boolean>

The condition the element must satisfy.

Returns

T

The element if it satisfies the specified condition; otherwise the default element.

Exceptions

InvalidOperationException

More than one element satisfies the condition in `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.ToArray<T> (ImmutableArray<T>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Copies the contents of this array to a mutable array.

C#

```
public static T[] ToArray<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray);
```

Type Parameters

T

The type of element contained by the collection.

Parameters

immutableArray ImmutableArray<T>

The immutable array to copy into a mutable one.

Returns

T[]

The newly instantiated array.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ImmutableArrayExtensions.ToDictionary Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Overloads

Expand table

ToDictionary<TKey,T>(ImmutableArray<T>, Func<T,TKey>)	Creates a dictionary based on the contents of this array.
ToDictionary<TKey,T>(ImmutableArray<T>, Func<T,TKey>, IEqualityComparer<TKey>)	Creates a dictionary based on the contents of this array.
ToDictionary<TKey,TElement,T>(ImmutableArray<T>, Func<T,TKey>, Func<T,TElement>, IEqualityComparer<TKey>)	Creates a dictionary based on the contents of this array.
ToDictionary<TKey,TElement,T>(ImmutableArray<T>, Func<T,TKey>, Func<T,TElement>)	Creates a dictionary based on the contents of this array.

ToDictionary<TKey,T>(ImmutableArray<T>, Func<T,TKey>)

Source: ImmutableArrayExtensions.cs

Creates a dictionary based on the contents of this array.

C#

```
public static System.Collections.Generic.Dictionary<TKey,T>
ToDictionary<TKey,T> (this System.Collections.Immutable.ImmutableArray<T>
immutableArray, Func<T,TKey> keySelector);
```

Type Parameters

`TKey`

The type of the key.

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to create a dictionary from.

`keySelector` Func<T,TKey>

The key selector.

Returns

Dictionary<TKey,T>

The newly initialized dictionary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

ToDictionary<TKey,T>(ImmutableArray<T>, Func<T,TKey>, IEqualityComparer<TKey>)

Source: ImmutableArrayExtensions.cs

Creates a dictionary based on the contents of this array.

```C#
public static System.Collections.Generic.Dictionary<TKey,T>
ToDictionary<TKey,T> (this System.Collections.Immutable.ImmutableArray<T>
immutableArray, Func<T,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TKey`

The type of the key.

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to create a dictionary from.

`keySelector` Func<T,TKey>

The key selector.

`comparer` IEqualityComparer<TKey>

The comparer to initialize the dictionary with.

Returns

Dictionary<TKey,T>

The newly initialized dictionary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

ToDictionary<TKey,TElement,T> (ImmutableArray<T>, Func<T,TKey>, Func<T,TElement>, IEqualityComparer<TKey>)

Source: ImmutableArrayExtensions.cs

Creates a dictionary based on the contents of this array.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TKey,TElement,T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,TKey> keySelector, Func<T,TElement> elementSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TKey`

The type of the key.

`TElement`

The type of the element.

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to create a dictionary from.

`keySelector` Func<T,TKey>

The key selector.

`elementSelector` Func<T,TElement>

The element selector.

`comparer` IEqualityComparer<TKey>

The comparer to initialize the dictionary with.

Returns

Dictionary<TKey,TElement>

The newly initialized dictionary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

ToDictionary<TKey,TElement,T> (ImmutableArray<T>, Func<T,TKey>, Func<T,TElement>)

Source: ImmutableArrayExtensions.cs

Creates a dictionary based on the contents of this array.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TKey,TElement,T> (this
```

```
System.Collections.Immutable.ImmutableArray<T> immutableArray,
Func<T,TKey> keySelector, Func<T,TElement> elementSelector);
```

Type Parameters

`TKey`

The type of the key.

`TElement`

The type of the element.

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to create a dictionary from.

`keySelector` Func<T,TKey>

The key selector.

`elementSelector` Func<T,TElement>

The element selector.

Returns

Dictionary<TKey,TElement>

The newly initialized dictionary.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ImmutableArrayExtensions.Where<T> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Collections.Immutable.dll

Package: System.Collections.Immutable v9.0.0-preview.7.24405.7

Source: ImmutableArrayExtensions.cs

Filters a sequence of values based on a predicate.

C#

```
public static System.Collections.Generic.IEnumerable<T> Where<T> (this
System.Collections.Immutable.ImmutableArray<T> immutableArray, Func<T,bool>
predicate);
```

Type Parameters

`T`

The type of element contained by the collection.

Parameters

`immutableArray` ImmutableArray<T>

The array to filter.

`predicate` Func<T,Boolean>

The condition to use for filtering the array content.

Returns

IEnumerable<T>

Returns IEnumerable<T> that contains elements that meet the condition.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IOrderedEnumerable<TElement> Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Represents a sorted sequence.

C#

```
public interface IOrderedEnumerable<out TElement> :
System.Collections.Generic.IEnumerable<out TElement>
```

Type Parameters

`TElement`

The type of the elements of the sequence.

Implements IEnumerable<TElement> , IEnumerable

Examples

The following example demonstrates how to perform a primary and secondary ordering on an array of strings. It also demonstrates that the resulting IOrderedEnumerable<TElement> is enumerable.

C#

```
// Create an array of strings to sort.
string[] fruits = { "apricot", "orange", "banana", "mango", "apple",
"grape", "strawberry" };

// Sort the strings first by their length and then alphabetically
// by passing the identity selector function.
IOrderedEnumerable<string> sortedFruits1 =
    fruits.OrderBy(fruit => fruit.Length).ThenBy(fruit => fruit);

// Output the resulting sequence of strings.
foreach (string fruit in sortedFruits1)
```

```
    Console.WriteLine(fruit);

// This code produces the following output:
//
// apple
// grape
// mango
// banana
// orange
// apricot
// strawberry
```

Remarks

This type is enumerable because it inherits from IEnumerable<T>.

The extension methods ThenBy and ThenByDescending operate on objects of type IOrderedEnumerable<TElement>. An object of type IOrderedEnumerable<TElement> can be obtained by calling one of the primary sort methods, OrderBy or OrderByDescending, which return an IOrderedEnumerable<TElement>. ThenBy and ThenByDescending, the subordinate sort methods, in turn also return an object of type IOrderedEnumerable<TElement>. This design allows for any number of consecutive calls to ThenBy or ThenByDescending, where each call performs a subordinate ordering on the sorted data returned from the previous call.

Methods

Expand table

CreateOrderedEnumerable<TKey> (Func<TElement,TKey>, IComparer<TKey>, Boolean)	Performs a subsequent ordering on the elements of an IOrderedEnumerable<TElement> according to a key.
GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.

ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.

CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each

	element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.

Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a

	specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for

	each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default

	equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.

LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.

Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.

MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a

	sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.

OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.

SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ThenBy<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(IOrderedEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

ThenByDescending<TSource,TKey>(IOrdered Enumerable<TSource>, Func<TSource,TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(IOrdered Enumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEquality Comparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an

	IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEquality	Produces the set union of two sequences according to a

Comparer<TKey>)	specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.

DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

IOrderedEnumerable<TElement>.CreateOrderedEnumerable<TKey> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Performs a subsequent ordering on the elements of an IOrderedEnumerable<TElement> according to a key.

C#

```
public System.Linq.IOrderedEnumerable<out TElement>
CreateOrderedEnumerable<TKey> (Func<out TElement,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer, bool descending);
```

Type Parameters

TKey

The type of the key produced by `keySelector`.

Parameters

keySelector Func<TElement,TKey>

The Func<T,TResult> used to extract the key for each element.

comparer IComparer<TKey>

The IComparer<T> used to compare keys for placement in the returned sequence.

descending Boolean

`true` to sort the elements in descending order; `false` to sort the elements in ascending order.

Returns

IOrderedEnumerable<TElement>

An IOrderedEnumerable<TElement> whose elements are sorted according to a key.

Examples

The following code example demonstrates how to use CreateOrderedEnumerable to perform a secondary ordering on an IOrderedEnumerable<TElement>.

C#

```
// Create an array of strings to sort.
string[] fruits = { "apricot", "orange", "banana", "mango", "apple",
"grape", "strawberry" };
// First sort the strings by their length.
IOrderedEnumerable<string> sortedFruits2 =
    fruits.OrderBy(fruit => fruit.Length);
// Secondarily sort the strings alphabetically, using the default comparer.
IOrderedEnumerable<string> sortedFruits3 =
    sortedFruits2.CreateOrderedEnumerable<string>(
        fruit => fruit,
        Comparer<string>.Default, false);

// Output the resulting sequence of strings.
foreach (string fruit in sortedFruits3)
    Console.WriteLine(fruit);

// This code produces the following output:
//
// apple
// grape
// mango
// banana
// orange
// apricot
// strawberry
```

Remarks

The functionality provided by this method is like that provided by ThenBy or ThenByDescending, depending on whether `descending` is `true` or `false`. They both perform a subordinate ordering of an already sorted sequence of type IOrderedEnumerable<TElement>.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IOrderedQueryable Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Represents the result of a sorting operation.

C#

```
public interface IOrderedQueryable : System.Collections.IEnumerable,
System.Linq.IQueryable
```

Derived System.Data.Objects.ObjectQuery
System.Data.Objects.ObjectQuery<T>
System.Linq.EnumerableQuery<T>
System.Linq.IOrderedQueryable<T>

Implements IEnumerable , IQueryable

Remarks

The IOrderedQueryable interface is intended for implementation by query providers.

This interface represents the result of a sorting query that calls the method(s) OrderBy, OrderByDescending, ThenBy or ThenByDescending. When CreateQuery is called and passed an expression tree that represents a sorting query, the resulting IQueryable object must be of a type that implements IOrderedQueryable.

For more information about how to create your own LINQ provider, see LINQ: Building an IQueryable Provider.

Properties

Expand table

Element Type	Gets the type of the element(s) that are returned when the expression tree associated with this instance of IQueryable is executed. (Inherited from IQueryable)
Expression	Gets the expression tree that is associated with the instance of IQueryable. (Inherited from IQueryable)
Provider	Gets the query provider that is associated with this data source. (Inherited from IQueryable)

Methods

Expand table

GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
Cast<TResult>(IQueryable)	Converts the elements of an IQueryable to the specified type.
OfType<TResult>(IQueryable)	Filters the elements of an IQueryable based on a specified type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Queryable
- IQueryable<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IOrderedQueryable<T> Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Represents the result of a sorting operation.

C#

```
public interface IOrderedQueryable<out T> :
System.Collections.Generic.IEnumerable<out T>,
System.Linq.IOrderedQueryable, System.Linq.IQueryable<out T>
```

Type Parameters

T

The type of the content of the data source.
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.

Derived System.Data.Objects.ObjectQuery<T>
System.Linq.EnumerableQuery<T>

Implements IEnumerable<T> , IEnumerable , IOrderedQueryable , IQueryable , IQueryable<T>

Remarks

The IOrderedQueryable<T> interface is intended for implementation by query providers.

This interface represents the result of a sorting query that calls the method(s) OrderBy, OrderByDescending, ThenBy or ThenByDescending. When CreateQuery<TElement> (Expression) is called and passed an expression tree that represents a sorting query, the

resulting IQueryable<T> object must be of a type that implements IOrderedQueryable<T>.

For more information about how to create your own LINQ provider, see LINQ: Building an IQueryable Provider.

Properties

Expand table

Element Type	Gets the type of the element(s) that are returned when the expression tree associated with this instance of IQueryable is executed. (Inherited from IQueryable)
Expression	Gets the expression tree that is associated with the instance of IQueryable. (Inherited from IQueryable)
Provider	Gets the query provider that is associated with this data source. (Inherited from IQueryable)

Methods

Expand table

GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified

	key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEquality Comparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEquality Comparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and

	produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.

ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function

	is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are

	obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified

	sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default

	value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the

	sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value

	from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>,	Correlates the elements of two sequences based on

Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic

	sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each

	element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the

	minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.

OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an

	IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an

	exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such

	element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on

	each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform

	function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.

ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization

	of a query.
Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether all the elements of a sequence satisfy a condition.
Any<TSource>(IQueryable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` appended at the end.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a

	generic IQueryable<T>.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the average of a sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the average of a sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the average of a sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the average of a sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the average of a sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the average of a sequence of nullable Double values that is obtained by invoking a projection function on each

	element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the average of a sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the average of a sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the average of a sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the average of a sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Cast<TResult>(IQueryable)	Converts the elements of an IQueryable to the specified type.
Chunk<TSource>(IQueryable<TSource>, Int32)	Split the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IQueryable<TSource>, TSource)	Determines whether a sequence contains a

	specified element by using the default equality comparer.
Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IQueryable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the number of elements in the specified sequence that satisfies a condition.
DefaultIfEmpty<TSource>(IQueryable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IQueryable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns distinct elements from a sequence according to

	a specified key selector function.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
ElementAt<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set difference of two

	sequences according to a specified key selector function.
First<TSource>(IQueryable<TSource>)	Returns the first element of a sequence.
First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IQueryable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>)	Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>,	Groups the elements of a sequence according to a specified key

Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>)	selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>)	Correlates the elements of two sequences based on key equality and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set intersection of two sequences according to a specified key selector function.

IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IQueryable<TSource>)	Returns the last element in a sequence.
Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IQueryable<TSource>)	Returns the last element in a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LongCount<TSource>(IQueryable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns an Int64 that represents the number of elements in a sequence that satisfy a condition.
Max<TSource>(IQueryable<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the maximum resulting value.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
Min<TSource>(IQueryable<TSource>)	Returns the minimum value of a generic IQueryable<T>.

Min<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T>.
Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the minimum resulting value.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
OfType<TResult>(IQueryable)	Filters the elements of an IQueryable based on a specified type.
Order<T>(IQueryable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in

	descending order according to a key.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IQueryable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` prepended at the beginning.
Reverse<TSource>(IQueryable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one

	sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> and invokes a result selector function on each element therein. The resulting values from each intermediate sequence are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> that incorporates the index of the source element that produced it. A result selector function is invoked on each element of each intermediate sequence, and the resulting values are combined into a single, one-dimensional sequence and returned.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by using the default equality comparer to compare elements.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by using a specified IEqualityComparer<T> to compare elements.

Single<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the only element of a sequence that satisfies a specified condition or a

	default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IQueryable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the elements from `source` with the last `count` elements of the source queryable sequence omitted.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the sum of the sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the sum of the sequence of Double values that is obtained by invoking a

	projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the sum of the sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the sum of the sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the sum of the sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the sum of the sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the sum of the sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the sum of the sequence of nullable Int64 values that is obtained by

	invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the sum of the sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the sum of the sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Take<TSource>(IQueryable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IQueryable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the last `count` elements from `source`.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.
ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Filters a sequence of values based on a

	predicate.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IQueryable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IQueryable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IQueryable<TFirst>, IEnumerable<TSecond>, Expression<Func<TFirst,TSecond,TResult>>)	Merges two sequences by using the specified predicate function.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document

	in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Queryable
- IQueryable<T>

IQueryable Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Provides functionality to evaluate queries against a specific data source wherein the type of the data is not specified.

C#

```
public interface IQueryable : System.Collections.IEnumerable
```

Derived System.Data.Linq.ITable
System.Data.Linq.ITable<TEntity>
System.Data.Linq.Table<TEntity>
System.Data.Objects.IObjectSet<TEntity>
System.Data.Objects.ObjectQuery
More...

Implements IEnumerable

Remarks

The IQueryable interface is intended for implementation by query providers. It is only supposed to be implemented by providers that also implement IQueryable<T>. If the provider does not also implement IQueryable<T>, the standard query operators cannot be used on the provider's data source.

The IQueryable interface inherits the IEnumerable interface so that if it represents a query, the results of that query can be enumerated. Enumeration causes the expression tree associated with an IQueryable object to be executed. The definition of "executing an expression tree" is specific to a query provider. For example, it may involve translating the expression tree to an appropriate query language for the underlying data source. Queries that do not return enumerable results are executed when the Execute method is called.

For more information about how to create your own LINQ provider, see LINQ: Building an IQueryable Provider.

Properties

Expand table

Element Type	Gets the type of the element(s) that are returned when the expression tree associated with this instance of IQueryable is executed.
Expression	Gets the expression tree that is associated with the instance of IQueryable.
Provider	Gets the query provider that is associated with this data source.

Methods

Expand table

GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
Cast<TResult>(IQueryable)	Converts the elements of an IQueryable to the specified type.
OfType<TResult>(IQueryable)	Filters the elements of an IQueryable based on a specified type.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Queryable
- IQueryable<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IQueryable.ElementType Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Gets the type of the element(s) that are returned when the expression tree associated with this instance of IQueryable is executed.

```
C#
public Type ElementType { get; }
```

Property Value

Type

A Type that represents the type of the element(s) that are returned when the expression tree associated with this object is executed.

Remarks

The ElementType property represents the "T" in `IQueryable<T>` or `IQueryable(Of T)`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IQueryable.Expression Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Gets the expression tree that is associated with the instance of IQueryable.

C#

```
public System.Linq.Expressions.Expression Expression { get; }
```

Property Value

Expression

The Expression that is associated with this instance of IQueryable.

Remarks

If an instance of IQueryable represents a LINQ query against a data source, the associated expression tree represents that query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IQueryable.Provider Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Gets the query provider that is associated with this data source.

C#

```
public System.Linq.IQueryProvider Provider { get; }
```

Property Value

IQueryProvider

The IQueryProvider that is associated with this data source.

Remarks

If an instance of IQueryable represents a LINQ query against a data source, the associated query provider is the provider that created the IQueryable instance.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IQueryable<T> Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Provides functionality to evaluate queries against a specific data source wherein the type of the data is known.

C#

```
public interface IQueryable<out T> :
System.Collections.Generic.IEnumerable<out T>, System.Linq.IQueryable
```

Type Parameters

T

The type of the data in the data source.
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.

Derived System.Data.Linq.ITable<TEntity>
System.Data.Linq.Table<TEntity>
System.Data.Objects.IObjectSet<TEntity>
System.Data.Objects.ObjectQuery<T>
System.Data.Objects.ObjectSet<TEntity>
More...

Implements IEnumerable<T> , IEnumerable , IQueryable

Remarks

The IQueryable<T> interface is intended for implementation by query providers.

This interface inherits the IEnumerable<T> interface so that if it represents a query, the results of that query can be enumerated. Enumeration forces the expression tree

associated with an IQueryable<T> object to be executed. Queries that do not return enumerable results are executed when the Execute<TResult>(Expression) method is called.

The definition of "executing an expression tree" is specific to a query provider. For example, it may involve translating the expression tree to a query language appropriate for an underlying data source.

The IQueryable<T> interface enables queries to be polymorphic. That is, because a query against an `IQueryable` data source is represented as an expression tree, it can be executed against different types of data sources.

The `static` (`Shared` in Visual Basic) methods defined in the class Queryable (except for AsQueryable, ThenBy, and ThenByDescending) extend objects of types that implement the IQueryable<T> interface.

For more information about how to create your own LINQ provider, see LINQ: Building an IQueryable Provider.

Properties

Expand table

Element Type	Gets the type of the element(s) that are returned when the expression tree associated with this instance of IQueryable is executed. (Inherited from IQueryable)
Expression	Gets the expression tree that is associated with the instance of IQueryable. (Inherited from IQueryable)
Provider	Gets the query provider that is associated with this data source. (Inherited from IQueryable)

Methods

Expand table

GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary

	of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.

ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.

Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on

	each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform

	function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the

	specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.

ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.

FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are

	projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.

Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value

	if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the

	maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.

MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence

	and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.

OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a

	new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.

SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The

	element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each

	element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last

	`count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector

	function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer

	and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.

Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether all the elements of a sequence satisfy a condition.
Any<TSource>(IQueryable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence

	that contains the elements from `source` plus the specified `element` appended at the end.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the average of a sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the average of a sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the average of a sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the average of a sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the average of a sequence of nullable Decimal values that is obtained by

	invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the average of a sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the average of a sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the average of a sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the average of a sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the average of a sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Cast<TResult>(IQueryable)	Converts the elements of an IQueryable to the

	specified type.
Chunk<TSource>(IQueryable<TSource>, Int32)	Split the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IQueryable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IQueryable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the number of elements in the specified sequence that satisfies a condition.
DefaultIfEmpty<TSource>(IQueryable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IQueryable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.

Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
ElementAt<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified

	IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IQueryable<TSource>)	Returns the first element of a sequence.
First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IQueryable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Groups the elements of a sequence according

	to a specified key selector function.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>)	Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer.

GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>)	Correlates the elements of two sequences based on key equality and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality

	comparer to compare values.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IQueryable<TSource>)	Returns the last element in a sequence.
Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IQueryable<TSource>)	Returns the last element in a sequence, or a default value if the

	sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LongCount<TSource>(IQueryable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns an Int64 that represents the number of elements in a sequence that satisfy a condition.
Max<TSource>(IQueryable<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the maximum resulting value.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the maximum value in a generic

	IQueryable<T> according to a specified key selector function.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
Min<TSource>(IQueryable<TSource>)	Returns the minimum value of a generic IQueryable<T>.
Min<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T>.
Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the minimum resulting value.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
OfType<TResult>(IQueryable)	Filters the elements of an IQueryable based on a specified type.
Order<T>(IQueryable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending

	order.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IQueryable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` prepended at the beginning.
Reverse<TSource>(IQueryable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>)	Projects each element of a sequence into a new form by

	incorporating the element's index.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> and invokes a result selector function on each element therein. The resulting values from each intermediate sequence are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> that incorporates the index of the source element that produced it. A result selector function is invoked on each element of each intermediate sequence, and the resulting values are combined into a single, one-dimensional sequence and returned.

SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by using the default equality comparer to compare elements.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by using a specified IEqualityComparer<T> to compare elements.
Single<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IQueryable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the elements from `source` with the last `count` elements of the source queryable sequence omitted.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The

	element's index is used in the logic of the predicate function.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the sum of the sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the sum of the sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the sum of the sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the sum of the sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the sum of the sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the sum of the sequence of nullable Double values that is obtained by invoking a projection function on each

	element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the sum of the sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the sum of the sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the sum of the sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the sum of the sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Take<TSource>(IQueryable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IQueryable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the last

	`count` elements from `source`.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Filters a sequence of values based on a predicate.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IQueryable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements

	from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IQueryable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IQueryable<TFirst>, IEnumerable<TSecond>, Expression<Func<TFirst,TSecond,TResult>>)	Merges two sequences by using the specified predicate function.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a

	matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Queryable

IQueryProvider Interface

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Defines methods to create and execute queries that are described by an IQueryable object.

C#

```
public interface IQueryProvider
```

Derived System.Data.Linq.Table<TEntity>
System.Linq.EnumerableQuery<T>

Remarks

The IQueryProvider interface is intended for implementation by query providers.

For more information about how to create your own LINQ provider, see LINQ: Building an IQueryable Provider.

Methods

Expand table

CreateQuery(Expression)	Constructs an IQueryable object that can evaluate the query represented by a specified expression tree.
CreateQuery<TElement>(Expression)	Constructs an IQueryable<T> object that can evaluate the query represented by a specified expression tree.
Execute(Expression)	Executes the query represented by a specified expression tree.
Execute<TResult>(Expression)	Executes the strongly-typed query represented by a specified expression tree.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

See also

- Queryable
- IQueryable<T>

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

IQueryProvider.CreateQuery Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Overloads

Expand table

CreateQuery(Expression)	Constructs an IQueryable object that can evaluate the query represented by a specified expression tree.
CreateQuery<TElement>(Expression)	Constructs an IQueryable<T> object that can evaluate the query represented by a specified expression tree.

CreateQuery(Expression)

Constructs an IQueryable object that can evaluate the query represented by a specified expression tree.

C#

```
public System.Linq.IQueryable CreateQuery
(System.Linq.Expressions.Expression expression);
```

Parameters

`expression` Expression

An expression tree that represents a LINQ query.

Returns

IQueryable

An IQueryable that can evaluate the query represented by the specified expression tree.

Remarks

> ⓘ **Note**
>
> The Expression property of the returned IQueryable object is equal to `expression`.

The CreateQuery method is used to create new IQueryable objects, given an expression tree. The query that is represented by the returned object is associated with a specific LINQ provider.

Several of the standard query operator methods defined in Queryable, such as OfType and Cast, call this method. They pass it a MethodCallExpression that represents a LINQ query.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

CreateQuery<TElement>(Expression)

Constructs an IQueryable<T> object that can evaluate the query represented by a specified expression tree.

C#

```
public System.Linq.IQueryable<TElement> CreateQuery<TElement>
(System.Linq.Expressions.Expression expression);
```

Type Parameters

`TElement`

The type of the elements of the IQueryable<T> that is returned.

Parameters

`expression` Expression

An expression tree that represents a LINQ query.

Returns

IQueryable<TElement>

An IQueryable<T> that can evaluate the query represented by the specified expression tree.

Remarks

> ⓘ **Note**
>
> The Expression property of the returned IQueryable<T> object is equal to `expression`.

The CreateQuery method is used to create new IQueryable<T> objects, given an expression tree. The query that is represented by the returned object is associated with a specific LINQ provider.

Most of the Queryable standard query operator methods that return enumerable results call this method. They pass it a MethodCallExpression that represents a LINQ query.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	Versions
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

IQueryProvider.Execute Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Expressions.dll

Overloads

Expand table

Execute(Expression)	Executes the query represented by a specified expression tree.
Execute<TResult>(Expression)	Executes the strongly-typed query represented by a specified expression tree.

Execute(Expression)

Executes the query represented by a specified expression tree.

C#

```
public object? Execute (System.Linq.Expressions.Expression expression);
```

Parameters

`expression` Expression

An expression tree that represents a LINQ query.

Returns

Object

The value that results from executing the specified query.

Remarks

The Execute method executes queries that return a single value (instead of an enumerable sequence of values). Expression trees that represent queries that return enumerable results are executed when their associated IQueryable object is enumerated.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Execute<TResult>(Expression)

Executes the strongly-typed query represented by a specified expression tree.

C#

```
public TResult Execute<TResult> (System.Linq.Expressions.Expression
expression);
```

Type Parameters

`TResult`

The type of the value that results from executing the query.

Parameters

`expression` Expression

An expression tree that represents a LINQ query.

Returns

TResult

The value that results from executing the specified query.

Remarks

The Execute method executes queries that return a single value (instead of an enumerable sequence of values). Expression trees that represent queries that return enumerable results are executed when the IQueryable<T> object that contains the expression tree is enumerated.

The Queryable standard query operator methods that return singleton results call Execute. They pass it a MethodCallExpression that represents a LINQ query.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lookup<TKey,TElement> Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.cs

Represents a collection of keys each mapped to one or more values.

C#

```
public class Lookup<TKey,TElement> :
System.Collections.Generic.IEnumerable<System.Linq.IGrouping<TKey,TElement>>
, System.Linq.ILookup<TKey,TElement>
```

Type Parameters

TKey

The type of the keys in the Lookup<TKey,TElement>.

TElement

The type of the elements of each IEnumerable<T> value in the Lookup<TKey,TElement>.

Inheritance Object → Lookup<TKey,TElement>

Implements IEnumerable<IGrouping<TKey,TElement>> , IEnumerable , ILookup<TKey,TElement>

Examples

The following example creates a Lookup<TKey,TElement> from a collection of objects. It then enumerates the Lookup<TKey,TElement> and outputs each key and each value in the key's associated collection of values. It also demonstrates how to use the properties Count and Item[] and the methods Contains and GetEnumerator.

C#

```
class Package
{
    public string Company;
    public double Weight;
    public long TrackingNumber;
}

public static void LookupExample()
{
    // Create a list of Packages to put into a Lookup data structure.
    List<Package> packages = new List<Package> { new Package { Company =
"Coho Vineyard", Weight = 25.2, TrackingNumber = 89453312L },
                                                 new Package { Company =
"Lucerne Publishing", Weight = 18.7, TrackingNumber = 89112755L },
                                                 new Package { Company =
"Wingtip Toys", Weight = 6.0, TrackingNumber = 299456122L },
                                                 new Package { Company =
"Contoso Pharmaceuticals", Weight = 9.3, TrackingNumber = 670053128L },
                                                 new Package { Company =
"Wide World Importers", Weight = 33.8, TrackingNumber = 4665518773L } };

    // Create a Lookup to organize the packages. Use the first character of
Company as the key value.
    // Select Company appended to TrackingNumber for each element value in
the Lookup.
    Lookup<char, string> lookup = (Lookup<char, string>)packages.ToLookup(p
=> p.Company[0],
                                                        p => p.Company + " " +
p.TrackingNumber);

    // Iterate through each IGrouping in the Lookup and output the contents.
    foreach (IGrouping<char, string> packageGroup in lookup)
    {
        // Print the key value of the IGrouping.
        Console.WriteLine(packageGroup.Key);
        // Iterate through each value in the IGrouping and print its value.
        foreach (string str in packageGroup)
            Console.WriteLine("    {0}", str);
    }

    // This code produces the following output:
    //
    // C
    //     Coho Vineyard 89453312
    //     Contoso Pharmaceuticals 670053128
    // L
    //     Lucerne Publishing 89112755
    // W
    //     Wingtip Toys 299456122
    //     Wide World Importers 4665518773

    // Get the number of key-collection pairs in the Lookup.
    int count = lookup.Count;
```

```
    // Select a collection of Packages by indexing directly into the Lookup.
    IEnumerable<string> cgroup = lookup['C'];

    // Output the results.
    Console.WriteLine("\nPackages that have a key of 'C':");
    foreach (string str in cgroup)
        Console.WriteLine(str);

    // This code produces the following output:
    //
    // Packages that have a key of 'C'
    // Coho Vineyard 89453312
    // Contoso Pharmaceuticals 670053128

    // Determine if there is a key with the value 'G' in the Lookup.
    bool hasG = lookup.Contains('G');
}
```

Remarks

A Lookup<TKey,TElement> resembles a Dictionary<TKey,TValue>. The difference is that a Dictionary<TKey,TValue> maps keys to single values, whereas a Lookup<TKey,TElement> maps keys to collections of values.

You can create an instance of a Lookup<TKey,TElement> by calling ToLookup on an object that implements IEnumerable<T>.

> **Note**
>
> There is no public constructor to create a new instance of a Lookup<TKey,TElement>. Additionally, Lookup<TKey,TElement> objects are immutable, that is, you cannot add or remove elements or keys from a Lookup<TKey,TElement> object after it has been created.

Properties

Expand table

Count	Gets the number of key/value collection pairs in the Lookup<TKey,TElement>.
Item[TKey]	Gets the collection of values indexed by the specified key.

Methods

Expand table

ApplyResultSelector<TResult> (Func<TKey,IEnumerable<TElement>,TResult>)	Applies a transform function to each key and its associated values and returns the results.
Contains(TKey)	Determines whether a specified key is in the Lookup<TKey,TElement>.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetEnumerator()	Returns a generic enumerator that iterates through the Lookup<TKey,TElement>.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Explicit Interface Implementations

Expand table

IEnumerable.GetEnumerator()	Returns an enumerator that iterates through the Lookup<TKey,TElement>. This class cannot be inherited.

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T>

	according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses

	the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T>

	object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.

Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the

	default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.

ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.

GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if

	the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a

	sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according

	to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order

	by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>,	Projects each element of a sequence to an IEnumerable<T>, flattens the

Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a

	default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each

	element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function

	on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.

ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer

	and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
AsParallel(IEnumerable)	Enables parallelization of a query.

AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the

	source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Lookup<TKey,TElement>.Count Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.cs

Gets the number of key/value collection pairs in the Lookup<TKey,TElement>.

C#

```
public int Count { get; }
```

Property Value

Int32

The number of key/value collection pairs in the Lookup<TKey,TElement>.

Implements

Count

Examples

The following example demonstrates how to use Count to determine the number of key/value collection pairs in a Lookup<TKey,TElement>. This code example is part of a larger example provided for the Lookup<TKey,TElement> class.

C#

```
// Get the number of key-collection pairs in the Lookup.
int count = lookup.Count;
```

Remarks

The value of the Count property does not change because items cannot be added to or removed from a Lookup<TKey,TElement> object after it has been created.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lookup<TKey,TElement>.Item[TKey] Property

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.cs

Gets the collection of values indexed by the specified key.

C#

```
public System.Collections.Generic.IEnumerable<TElement> this[TKey key] {
get; }
```

Parameters

key TKey

The key of the desired collection of values.

Property Value

IEnumerable<TElement>

The collection of values indexed by the specified key.

Implements

Item[TKey]

Examples

The following example demonstrates how to use Item[] to index directly into a Lookup<TKey,TElement>. This code example is part of a larger example provided for the Lookup<TKey,TElement> class.

C#

```
// Select a collection of Packages by indexing directly into the Lookup.
IEnumerable<string> cgroup = lookup['C'];
```

Remarks

This indexed property provides the ability to index a specific collection of values in the Lookup<TKey,TElement> by using the following syntax: `myLookup[key]` in Visual C# or `myLookup(key)` in Visual Basic. If the `key` is not found in the collection, an empty sequence is returned.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lookup<TKey,TElement>.ApplyResultSelector<TResult> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.SpeedOpt.cs

Applies a transform function to each key and its associated values and returns the results.

C#

```
public System.Collections.Generic.IEnumerable<TResult>
ApplyResultSelector<TResult>
(Func<TKey,System.Collections.Generic.IEnumerable<TElement>,TResult>
resultSelector);
```

Type Parameters

`TResult`

The type of the result values produced by `resultSelector`.

Parameters

`resultSelector` Func<TKey,IEnumerable<TElement>,TResult>

A function to project a result value from each key and its associated values.

Returns

IEnumerable<TResult>

A collection that contains one value for each key/value collection pair in the Lookup<TKey,TElement>.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lookup<TKey,TElement>.Contains(TKey) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.cs

Determines whether a specified key is in the Lookup<TKey,TElement>.

C#

```
public bool Contains (TKey key);
```

Parameters

key TKey

The key to find in the Lookup<TKey,TElement>.

Returns

Boolean

`true` if `key` is in the Lookup<TKey,TElement>; otherwise, `false`.

Implements

Contains(TKey)

Examples

The following example demonstrates how to use Contains to determine whether a Lookup<TKey,TElement> contains a specified key. This code example is part of a larger example provided for the Lookup<TKey,TElement> class.

C#

```
// Determine if there is a key with the value 'G' in the Lookup.
bool hasG = lookup.Contains('G');
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lookup<TKey,TElement>.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.SpeedOpt.cs

Returns a generic enumerator that iterates through the Lookup<TKey,TElement>.

C#

```
public
System.Collections.Generic.IEnumerator<System.Linq.IGrouping<TKey,TElement>>
GetEnumerator ();
```

Returns

IEnumerator<IGrouping<TKey,TElement>>

An enumerator for the Lookup<TKey,TElement>.

Implements

GetEnumerator()

Examples

The following example demonstrates how to use GetEnumerator to iterate through the keys and values of a Lookup<TKey,TElement>. This code example is part of a larger example provided for the Lookup<TKey,TElement> class.

C#

```
// Iterate through each IGrouping in the Lookup and output the contents.
foreach (IGrouping<char, string> packageGroup in lookup)
{
    // Print the key value of the IGrouping.
    Console.WriteLine(packageGroup.Key);
```

```
    // Iterate through each value in the IGrouping and print its value.
    foreach (string str in packageGroup)
        Console.WriteLine("    {0}", str);
}
```

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Lookup<TKey,TElement>.IEnumerable.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.dll

Source: Lookup.cs

Returns an enumerator that iterates through the Lookup<TKey,TElement>. This class cannot be inherited.

```
C#
System.Collections.IEnumerator IEnumerable.GetEnumerator ();
```

Returns

IEnumerator

An enumerator for the Lookup<TKey,TElement>.

Implements

GetEnumerator()

Remarks

This member is an explicit interface member implementation. It can be used only when the Lookup<TKey,TElement> instance is cast to an IEnumerable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

OrderedParallelQuery<TSource> Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: OrderedParallelQuery.cs

Represents a sorted, parallel sequence.

C#

```
public class OrderedParallelQuery<TSource> :
System.Linq.ParallelQuery<TSource>
```

Type Parameters

TSource

The type of elements in the source collection.

Inheritance Object → ParallelQuery → ParallelQuery<TSource> → OrderedParallelQuery<TSource>

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetEnumerator()	Returns an enumerator that iterates through the sequence.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)

ToString()	Returns a string that represents the current object. (Inherited from Object)

Explicit Interface Implementations

Expand table

IEnumerable.GetEnumerator()	Returns an enumerator that iterates through the sequence. (Inherited from ParallelQuery)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>,	Enumerates and transforms a sequence, and produces an

Func<TSource,TValue>)	immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEquality Comparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEquality Comparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.

ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.

Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a

	specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.

ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a

	condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are

	compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.

LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the

	maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.

Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.

MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.

Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by

	using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception

	if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.

Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key

	selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.
TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.

Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
Aggregate<TSource>(ParallelQuery<TSource>, Func<TSource,TSource,TSource>)	Applies in parallel an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate,	Applies in parallel an accumulator function over a

Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate, TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	sequence. This overload is not available in the sequential implementation.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, Func<TAccumulate>, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate, TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
All<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether all elements of a sequence satisfy a condition.
Any<TSource>(ParallelQuery<TSource>)	Determines whether a parallel sequence contains any elements.
Any<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether any element of a sequence satisfies a condition.
AsEnumerable<TSource>(ParallelQuery<TSource>)	Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.
AsOrdered(ParallelQuery)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on non-generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsOrdered<TSource>(ParallelQuery<TSource>)	Enables treatment of a data source as if it were ordered, overriding the default of

	unordered. AsOrdered may only be invoked on generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsSequential<TSource>(ParallelQuery<TSource>)	Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.
AsUnordered<TSource>(ParallelQuery<TSource>)	Allows an intermediate query to be treated as if no ordering is implied among the elements.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(ParallelQuery)	Converts the elements of a ParallelQuery to the specified type.
Concat<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.**

	This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Concat<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Concatenates two parallel sequences.
Contains<TSource>(ParallelQuery<TSource>, TSource)	Determines in parallel whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(ParallelQuery<TSource>, TSource, IEqualityComparer<TSource>)	Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(ParallelQuery<TSource>)	Returns the number of elements in a parallel sequence.
Count<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.
DefaultIfEmpty<TSource>(ParallelQuery<TSource>)	Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(ParallelQuery<TSource>, TSource)	Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(ParallelQuery<TSource>)	Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.
Distinct<TSource>(ParallelQuery<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a parallel sequence by

	using a specified IEqualityComparer<T> to compare values.
ElementAt<TSource>(ParallelQuery<TSource>, Int32)	Returns the element at a specified index in a parallel sequence.
ElementAtOrDefault<TSource>(ParallelQuery<TSource>, Int32)	Returns the element at a specified index in a parallel sequence or a default value if the index is out of range.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, Parallel Query<TSource>)	Produces the set difference of two parallel sequences by using the default equality comparer to compare values.
Except<TSource>(ParallelQuery<TSource>, Parallel Query<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two parallel sequences by using the specified IEqualityComparer<T> to compare values.
First<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence.
First<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element in a parallel sequence that satisfies a specified condition.
FirstOrDefault<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence, or a

	default value if the sequence contains no elements.
FirstOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.
ForAll<TSource>(ParallelQuery<TSource>, Action<TSource>)	Invokes in parallel the specified action for each element in the `source`.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified IComparer<T>.
GroupBy<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>,	Groups in parallel the elements of a sequence according to a specified key

Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The

	default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set intersection of two parallel sequences by using the default equality comparer to compare values.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two parallel sequences by using the specified IEqualityComparer{T} to compare values.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence.
Last<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a specified condition.
LastOrDefault<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.
LongCount<TSource>(ParallelQuery<TSource>)	Returns an Int64 that represents the total number of elements in a parallel sequence.
LongCount<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.

Max<TSource>(ParallelQuery<TSource>)	Returns the maximum value in a parallel sequence of values.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each

	element of a sequence and returns the maximum value.
Max<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Min<TSource>(ParallelQuery<TSource>)	Returns the minimum value in a parallel sequence of values.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
OfType<TResult>(ParallelQuery)	Filters the elements of a ParallelQuery based on a specified type.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
Reverse<TSource>(ParallelQuery<TSource>)	Inverts the order of the elements in a parallel sequence.
Select<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Projects in parallel each element of a sequence into a new form.
Select<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,TResult>)	Projects in parallel each element of a sequence into a new form by incorporating the element's index.

SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(Parallel Query<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(Parallel Query<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
SequenceEqual<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws

	NotSupportedException when called.
SequenceEqual<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Determines whether two parallel sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Determines whether two parallel sequences are equal by comparing their elements by using a specified IEqualityComparer{T}.
Single<TSource>(ParallelQuery<TSource>)	Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(ParallelQuery<TSource>)	Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(ParallelQuery<TSource>, Int32)	Bypasses a specified number of elements in a parallel sequence and then returns the remaining elements.

SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(ParallelQuery<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a parallel sequence.
TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns elements from a parallel sequence as long as a specified condition is true.
TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a parallel sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

ThenBy<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.
ThenByDescending<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
ToArray<TSource>(ParallelQuery<TSource>)	Creates an array from a ParallelQuery<TSource>.
ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function.
ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to specified key selector and element selector functions.

ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function, a comparer, and an element selector function.
ToList<TSource>(ParallelQuery<TSource>)	Creates a List<T> from an ParallelQuery<TSource>.
ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function.
ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function, a comparer and an element selector function.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws

	NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set union of two parallel sequences by using the default equality comparer.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set union of two parallel sequences by using a specified IEqualityComparer{T}.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Filters in parallel a sequence of values based on a predicate.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken)	Sets the CancellationToken to associate with the query.
WithDegreeOfParallelism<TSource>(ParallelQuery<TSource>, Int32)	Sets the degree of parallelism to use in a query. Degree of parallelism is the maximum number of concurrently executing tasks that will be used to process the query.
WithExecutionMode<TSource>(ParallelQuery<TSource>, ParallelExecutionMode)	Sets the execution mode of the query.
WithMergeOptions<TSource>(ParallelQuery<TSource>, ParallelMergeOptions)	Sets the merge options for this query, which specify how the query will buffer output.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	**Obsolete.**

	This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, ParallelQuery<TSecond>, Func<TFirst,TSecond,TResult>)	Merges in parallel two sequences by using the specified predicate function.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.

Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- ParallelEnumerable
- Parallel LINQ (PLINQ)
- Order Preservation in PLINQ
- How to: Control Ordering in a PLINQ Query

OrderedParallelQuery<TSource>.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: OrderedParallelQuery.cs

Returns an enumerator that iterates through the sequence.

C#

```
public override System.Collections.Generic.IEnumerator<TSource>
GetEnumerator ();
```

Returns

IEnumerator<TSource>

An enumerator that iterates through the sequence.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- ParallelEnumerable
- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Provides a set of methods for querying objects that implement ParallelQuery{TSource}. This is the parallel equivalent of Enumerable.

C#

```
public static class ParallelEnumerable
```

Inheritance Object → ParallelEnumerable

Methods

Expand table

Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, Func<TAccumulate>, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.

Aggregate<TSource,TAccumulate>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource>(ParallelQuery<TSource>, Func<TSource,TSource,TSource>)	Applies in parallel an accumulator function over a sequence.
All<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether all elements of a sequence satisfy a condition.
Any<TSource>(ParallelQuery<TSource>)	Determines whether a parallel sequence contains any elements.
Any<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether any element of a sequence satisfies a condition.
AsEnumerable<TSource>(ParallelQuery<TSource>)	Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.
AsOrdered(ParallelQuery)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on non-generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsOrdered<TSource>(ParallelQuery<TSource>)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.

AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsParallel<TSource>(Partitioner<TSource>)	Enables parallelization of a query, as sourced by a custom partitioner that is responsible for splitting the input sequence into partitions.
AsSequential<TSource>(ParallelQuery<TSource>)	Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.
AsUnordered<TSource>(ParallelQuery<TSource>)	Allows an intermediate query to be treated as if no ordering is implied among the elements.
Average(ParallelQuery<Decimal>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Double>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Int32>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Int64>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Decimal>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Double>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Int32>>)	Computes in parallel the average of a sequence of values.

Average(ParallelQuery<Nullable<Int64>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Single>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Single>)	Computes in parallel the average of a sequence of values.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(ParallelQuery)	Converts the elements of a ParallelQuery to the specified type.
Concat<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Concat<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Concatenates two parallel sequences.
Contains<TSource>(ParallelQuery<TSource>, TSource)	Determines in parallel whether a sequence contains a specified element by using

	the default equality comparer.
Contains<TSource>(ParallelQuery<TSource>, TSource, IEqualityComparer<TSource>)	Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(ParallelQuery<TSource>)	Returns the number of elements in a parallel sequence.
Count<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.
DefaultIfEmpty<TSource>(ParallelQuery<TSource>)	Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(ParallelQuery<TSource>, TSource)	Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(ParallelQuery<TSource>)	Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.
Distinct<TSource>(ParallelQuery<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a parallel sequence by using a specified IEqualityComparer<T> to compare values.
ElementAt<TSource>(ParallelQuery<TSource>, Int32)	Returns the element at a specified index in a parallel sequence.
ElementAtOrDefault<TSource>(ParallelQuery<TSource>, Int32)	Returns the element at a specified index in a parallel sequence or a default value if the index is out of range.

Empty<TResult>()	Returns an empty ParallelQuery{TResult} that has the specified type argument.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set difference of two parallel sequences by using the default equality comparer to compare values.
Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two parallel sequences by using the specified IEqualityComparer<T> to compare values.
First<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence.
First<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element in a parallel sequence that satisfies a specified condition.
FirstOrDefault<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.

ForAll<TSource>(ParallelQuery<TSource>, Action<TSource>)	Invokes in parallel the specified action for each element in the `source`.
GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified IComparer<T>.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified

	IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set intersection of two parallel sequences by using the default equality comparer to compare values.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two parallel sequences by using the specified IEqualityComparer{T} to compare values.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>,	Correlates in parallel the elements of two sequences

Func<TOuter,TInner,TResult>)	based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence.
Last<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a specified condition.
LastOrDefault<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.
LongCount<TSource>(ParallelQuery<TSource>)	Returns an Int64 that represents the total number of elements in a parallel sequence.
LongCount<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.
Max(ParallelQuery<Decimal>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Double>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Int32>)	Returns the maximum value in a parallel sequence of values.

Max(ParallelQuery<Int64>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Decimal>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Double>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Int32>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Int64>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Single>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Single>)	Returns the maximum value in a parallel sequence of values.
Max<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>)	Returns the maximum value in a parallel sequence of values.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Min(ParallelQuery<Decimal>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Double>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Int32>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Int64>)	Returns the minimum value in a parallel sequence of values.

Min(ParallelQuery<Nullable<Decimal>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Double>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Int32>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Int64>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Single>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Single>)	Returns the minimum value in a parallel sequence of values.
Min<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>)	Returns the minimum value in a parallel sequence of values.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each

	element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
OfType<TResult>(ParallelQuery)	Filters the elements of a ParallelQuery based on a specified type.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in descending order according to a key.

OrderByDescending<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
Range(Int32, Int32)	Generates a parallel sequence of integral numbers within a specified range.
Repeat<TResult>(TResult, Int32)	Generates a parallel sequence that contains one repeated value.
Reverse<TSource>(ParallelQuery<TSource>)	Inverts the order of the elements in a parallel sequence.
Select<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,TResult>)	Projects in parallel each element of a sequence into a new form by incorporating the element's index.
Select<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Projects in parallel each element of a sequence into a new form.
SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T> and flattens

	the resulting sequences into one sequence.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SequenceEqual<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
SequenceEqual<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
SequenceEqual<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Determines whether two parallel sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Determines whether two parallel sequences are equal by comparing their elements by using a specified IEqualityComparer{T}.
Single<TSource>(ParallelQuery<TSource>)	Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition,

	and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(ParallelQuery<TSource>)	Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(ParallelQuery<TSource>, Int32)	Bypasses a specified number of elements in a parallel sequence and then returns the remaining elements.
SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum(ParallelQuery<Decimal>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Double>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Int32>)	Computes in parallel the sum of a sequence of values.

Sum(ParallelQuery<Int64>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Decimal>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Double>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Int32>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Int64>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Single>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Single>)	Computes in parallel the sum of a sequence of values.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the sum of the sequence of values

	that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(ParallelQuery<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a parallel sequence.
TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns elements from a parallel sequence as long as a specified condition is true.
TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a parallel sequence as long as

	a specified condition is true. The element's index is used in the logic of the predicate function.
ThenBy<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.
ThenByDescending<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
ToArray<TSource>(ParallelQuery<TSource>)	Creates an array from a ParallelQuery<TSource>.
ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function, a comparer, and an element selector function.
ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue>

	from a ParallelQuery<TSource> according to a specified key selector function.
ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
ToList<TSource>(ParallelQuery<TSource>)	Creates a List<T> from an ParallelQuery<TSource>.
ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function, a comparer and an element selector function.
ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function.
ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.**

	This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set union of two parallel sequences by using the default equality comparer.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set union of two parallel sequences by using a specified IEqualityComparer{T}.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Filters in parallel a sequence of values based on a predicate.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken)	Sets the CancellationToken to associate with the query.
WithDegreeOfParallelism<TSource>(ParallelQuery<TSource>, Int32)	Sets the degree of parallelism to use in a query. Degree of parallelism is the maximum number of concurrently executing tasks that will be used to process the query.
WithExecutionMode<TSource>(ParallelQuery<TSource>, ParallelExecutionMode)	Sets the execution mode of the query.
WithMergeOptions<TSource>(ParallelQuery<TSource>, ParallelMergeOptions)	Sets the merge options for this query, which specify how

	the query will buffer output.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	**Obsolete.** This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, ParallelQuery<TSecond>, Func<TFirst,TSecond,TResult>)	Merges in parallel two sequences by using the specified predicate function.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Aggregate Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Applies in parallel an accumulator function over a sequence.

Overloads

Expand table

Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, Func<TAccumulate>, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate, TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate, TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
Aggregate<TSource,TAccumulate>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource>(ParallelQuery<TSource>, Func<TSource,TSource,TSource>)	Applies in parallel an accumulator function over a sequence.

Aggregate<TSource,TAccumulate,TResult> (ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)

Source: ParallelEnumerable.cs

Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.

C#

```
public static TResult Aggregate<TSource,TAccumulate,TResult> (this
System.Linq.ParallelQuery<TSource> source, TAccumulate seed,
Func<TAccumulate,TSource,TAccumulate> func, Func<TAccumulate,TResult>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

`TResult`

The type of the resulting value.

Parameters

`source` ParallelQuery<TSource>

A sequence to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`func` Func<TAccumulate,TSource,TAccumulate>

An accumulator function to be invoked on each element.

`resultSelector` Func<TAccumulate,TResult>

A function to transform the final accumulator value into the result value.

Returns

TResult

The transformed final accumulator value.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `func` or `resultSelector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Aggregate<TSource,TAccumulate,TResult> (ParallelQuery<TSource>, Func<TAccumulate>, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate, TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)

Source: ParallelEnumerable.cs

Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.

C#

```
public static TResult Aggregate<TSource,TAccumulate,TResult> (this
System.Linq.ParallelQuery<TSource> source, Func<TAccumulate> seedFactory,
Func<TAccumulate,TSource,TAccumulate> updateAccumulatorFunc,
Func<TAccumulate,TAccumulate,TAccumulate> combineAccumulatorsFunc,
Func<TAccumulate,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

`TResult`

The type of the resulting value.

Parameters

`source` ParallelQuery<TSource>

A sequence to aggregate over.

`seedFactory` Func<TAccumulate>

A function that returns the initial accumulator value.

`updateAccumulatorFunc` Func<TAccumulate,TSource,TAccumulate>

An accumulator function to be invoked on each element in a partition.

`combineAccumulatorsFunc` Func<TAccumulate,TAccumulate,TAccumulate>

An accumulator function to be invoked on the yielded accumulator result from each partition.

`resultSelector` Func<TAccumulate,TResult>

A function to transform the final accumulator value into the result value.

Returns

TResult

The transformed final accumulator value.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `seedFactory` or `updateAccumulatorFunc` or `combineAccumulatorsFunc` or `resultSelector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

Remarks

This overload is specific to parallelized queries. A parallelized query may partition the data source sequence into several sub-sequences (partitions). The `updateAccumulatorFunc` is invoked on each element within partitions. Each partition then yields a single accumulated result. The `combineAccumulatorsFunc` is then invoked on the results of each partition to yield a single element. This element is then transformed by the `resultSelector` function.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Aggregate<TSource,TAccumulate,TResult> (ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate, TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)

Source: ParallelEnumerable.cs

Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.

C#

```
public static TResult Aggregate<TSource,TAccumulate,TResult> (this
System.Linq.ParallelQuery<TSource> source, TAccumulate seed,
Func<TAccumulate,TSource,TAccumulate> updateAccumulatorFunc,
Func<TAccumulate,TAccumulate,TAccumulate> combineAccumulatorsFunc,
Func<TAccumulate,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

`TResult`

The type of the resulting value.

Parameters

`source` ParallelQuery<TSource>

A sequence to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`updateAccumulatorFunc` Func<TAccumulate,TSource,TAccumulate>

An accumulator function to be invoked on each element in a partition.

`combineAccumulatorsFunc` Func<TAccumulate,TAccumulate,TAccumulate>

An accumulator function to be invoked on the yielded accumulator result from each partition.

`resultSelector` Func<TAccumulate,TResult>

A function to transform the final accumulator value into the result value.

Returns

TResult

The transformed final accumulator value.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `updateAccumulatorFunc` or `combineAccumulatorsFunc` or `resultSelector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

Remarks

This overload is specific to processing a parallelized query. A parallelized query may partition the data source sequence into several sub-sequences (partitions). The `updateAccumulatorFunc` is invoked on each element within partitions. Each partition then yields a single accumulated result. The `combineAccumulatorsFunc` is then invoked on the results of each partition to yield a single element. This element is then transformed by the `resultSelector` function.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Aggregate<TSource,TAccumulate>(ParallelQuery<TSource>, TAccumulate,

Func<TAccumulate,TSource,TAccumulate>)

Source: ParallelEnumerable.cs

Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.

```
C#

public static TAccumulate Aggregate<TSource,TAccumulate> (this
System.Linq.ParallelQuery<TSource> source, TAccumulate seed,
Func<TAccumulate,TSource,TAccumulate> func);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

Parameters

`source` ParallelQuery<TSource>

A sequence to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`func` Func<TAccumulate,TSource,TAccumulate>

An accumulator function to be invoked on each element.

Returns

TAccumulate

The final accumulator value.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `func` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Aggregate<TSource>(ParallelQuery<TSource>, Func<TSource,TSource,TSource>)

Source: ParallelEnumerable.cs

Applies in parallel an accumulator function over a sequence.

C#

```
public static TSource Aggregate<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TSource,TSource>
func);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence to aggregate over.

`func` Func<TSource,TSource,TSource>

An accumulator function to be invoked on each element.

Returns

TSource

The final accumulator value.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `func` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.All<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Determines in parallel whether all elements of a sequence satisfy a condition.

C#

```
public static bool All<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,bool> predicate);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

source ParallelQuery<TSource>

A sequence whose elements to apply the predicate to.

predicate Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Boolean

`true` if every element of the source sequence passes the test in the specified predicate, or if the sequence is empty; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Any Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Determines whether a parallel sequence contains any elements.

Overloads

Expand table

Any<TSource>(ParallelQuery<TSource>)	Determines whether a parallel sequence contains any elements.
Any<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether any element of a sequence satisfies a condition.

Any<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Determines whether a parallel sequence contains any elements.

C#

```
public static bool Any<TSource> (this System.Linq.ParallelQuery<TSource>
source);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

source ParallelQuery<TSource>

The sequence to check for emptiness.

Returns

Boolean

`true` if the source sequence contains any elements; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

This overload of the Any method causes a fully-buffered merge to be performed before the result is returned. The merge behavior can impact performance or cause an out of memory exception in large enumerables. To avoid the merge behavior, use the overload of Any that takes a Func parameter, or else use an alternative method such as FirstOrDefault.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Any<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Determines in parallel whether any element of a sequence satisfies a condition.

```
C#

public static bool Any<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence to whose elements the predicate will be applied.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Boolean

`true` if any elements in the source sequence pass the test in the specified predicate; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.AsEnumerable<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
AsEnumerable<TSource> (this System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to cast as IEnumerable<T>.

Returns

IEnumerable<TSource>

The input sequence typed as IEnumerable<T>.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.AsOrdered Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Overloads

Expand table

AsOrdered(ParallelQuery)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on non-generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsOrdered<TSource>(ParallelQuery<TSource>)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.

AsOrdered(ParallelQuery)

Source: ParallelEnumerable.cs

Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on non-generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.

C#

```
public static System.Linq.ParallelQuery AsOrdered (this
System.Linq.ParallelQuery source);
```

Parameters

`source` ParallelQuery

The input sequence.

Returns

ParallelQuery

The source sequence which will maintain the original ordering in the subsequent query operators.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

InvalidOperationException

thrown if `source` contains no elements

-or-

if `AsOrdered` is called midway through a query. It is allowed to be called immediately after AsParallel(IEnumerable), Range(Int32, Int32), or Repeat<TResult>(TResult, Int32).

-or-

`source` is not one of AsParallel(IEnumerable), Range(Int32, Int32), or Repeat<TResult>(TResult, Int32).

Remarks

A natural tension exists between performance and preserving order in parallel processing. By default, a parallelized query behaves as if the ordering of the results is arbitrary unless AsOrdered is applied or there is an explicit OrderBy operator in the query. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

AsOrdered<TSource> (ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.

C#

```
public static System.Linq.ParallelQuery<TSource> AsOrdered<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The input sequence.

Returns

ParallelQuery<TSource>

The source sequence which will maintain the original ordering in the subsequent query operators.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

InvalidOperationException

`source` contains no elements

-or-

`source` is not one of AsParallel, ParallelEnumerable.Range, or ParallelEnumerable.Repeat.

Remarks

A natural tension exists between performance and preserving order in parallel processing. By default, a parallelized query behaves as if the ordering of the results is arbitrary unless AsOrdered is applied or there is an explicit OrderBy operator in the query. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.AsParallel Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Enables parallelization of a query.

Overloads

Expand table

AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource> (Partitioner<TSource>)	Enables parallelization of a query, as sourced by a custom partitioner that is responsible for splitting the input sequence into partitions.
AsParallel<TSource> (IEnumerable<TSource>)	Enables parallelization of a query.

Remarks

This method binds the query to PLINQ. For more information, see Parallel LINQ (PLINQ).

AsParallel(IEnumerable)

Source: ParallelEnumerable.cs

Enables parallelization of a query.

C#

```
public static System.Linq.ParallelQuery AsParallel (this
System.Collections.IEnumerable source);
```

Parameters

`source` IEnumerable

An IEnumerable<T> to convert to a ParallelQuery.

Returns

ParallelQuery

The source as a ParallelQuery to bind to ParallelEnumerable extension methods.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

Remarks

The Cast operator can be used to convert a ParallelQuery to a ParallelQuery(T).

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

AsParallel<TSource>(Partitioner<TSource>)

Source: ParallelEnumerable.cs

Enables parallelization of a query, as sourced by a custom partitioner that is responsible for splitting the input sequence into partitions.

```C#
public static System.Linq.ParallelQuery<TSource> AsParallel<TSource>
(this System.Collections.Concurrent.Partitioner<TSource> source);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` Partitioner<TSource>

A partitioner over the input sequence.

Returns

ParallelQuery<TSource>

The `source` as a ParallelQuery to bind to ParallelEnumerable extension methods.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

Remarks

The source partitioner's GetOrderedPartitions method is used when ordering is enabled, whereas the partitioner's GetPartitions is used if ordering is not enabled (the default). The source partitioner's GetDynamicPartitions and GetDynamicOrderedPartitions are not used. For more information and examples, see Custom Partitioners for PLINQ and TPL.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

AsParallel<TSource>(IEnumerable<TSource>)

Source: ParallelEnumerable.cs

Enables parallelization of a query.

C#

```
public static System.Linq.ParallelQuery<TSource> AsParallel<TSource>
(this System.Collections.Generic.IEnumerable<TSource> source);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` IEnumerable<TSource>

An IEnumerable<T> to convert to a ParallelQuery<TSource>.

Returns

ParallelQuery<TSource>

The source as a ParallelQuery<TSource> to bind to ParallelEnumerable extension methods.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.AsSequential<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.

C#

```
public static System.Collections.Generic.IEnumerable<TSource>
AsSequential<TSource> (this System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

source ParallelQuery<TSource>

A ParallelQuery<TSource> to convert to an IEnumerable<T>.

Returns

IEnumerable<TSource>

The source as an IEnumerable<T> to bind to sequential extension methods.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

Remarks

For more information, see Order Preservation in PLINQ and How to: Control Ordering in a PLINQ Query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.AsUnordered<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Allows an intermediate query to be treated as if no ordering is implied among the elements.

C#

```
public static System.Linq.ParallelQuery<TSource> AsUnordered<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

source ParallelQuery<TSource>

The input sequence.

Returns

ParallelQuery<TSource>

The source sequence with arbitrary order.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

Remarks

AsUnordered may provide performance benefits when ordering is not required in a portion of a query. By default, PLINQ treats an input sequence as unordered unless OrderBy or AsOrdered is specified. However, if ordering was turned on and is no longer needed, then AsUnordered can be used to turn it off in mid-query, and this may result in performance benefits. The AsUnordered operator itself does not shuffle the source sequence; it simply removes the ordering requirement for subsequent operators. If the source is ordered, subsequent operators might keep that ordering if it is more efficient to do so. AsUnordered can be called anywhere in the query; it does not need to be called immediately after AsParallel. For more information, see Understanding Speedup in PLINQ and Order Preservation in PLINQ.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Average Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Computes in parallel the average of a sequence of values.

Overloads

Expand table

Average(ParallelQuery<Single>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Int64>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Int32>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Double>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Single>>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Int64>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Int32>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Double>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Decimal>)	Computes in parallel the average of a sequence of values.
Average(ParallelQuery<Nullable<Decimal>>)	Computes in parallel the average of a sequence of values.

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Average(ParallelQuery<Single>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

```
C#

public static float Average (this System.Linq.ParallelQuery<float>
source);
```

Parameters

`source` ParallelQuery<Single>

A sequence of values that are used to calculate an average.

Returns

Single

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Nullable<Int64>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

C#

```
public static double? Average (this System.Linq.ParallelQuery<long?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Int64>>

A sequence of values that are used to calculate an average.

Returns

Nullable<Double>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Nullable<Int32>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

C#

```
public static double? Average (this System.Linq.ParallelQuery<int?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Int32>>

A sequence of values that are used to calculate an average.

Returns

Nullable<Double>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Average(ParallelQuery<Nullable<Double>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

```
C#

public static double? Average (this System.Linq.ParallelQuery<double?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Double>>

The source sequence.

Returns

Nullable<Double>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is `null`.

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Nullable<Single>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

C#

```
public static float? Average (this System.Linq.ParallelQuery<float?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Single>>

A sequence of values that are used to calculate an average.

Returns

Nullable<Single>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Int64>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

C#

```
public static double Average (this System.Linq.ParallelQuery<long>
source);
```

Parameters

`source` ParallelQuery<Int64>

A sequence of values that are used to calculate an average.

Returns

Double

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Int32>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

```
C#

public static double Average (this System.Linq.ParallelQuery<int>
source);
```

Parameters

`source` ParallelQuery<Int32>

A sequence of values that are used to calculate an average.

Returns

Double

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Double>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

C#

```
public static double Average (this System.Linq.ParallelQuery<double>
source);
```

Parameters

`source` ParallelQuery<Double>

A sequence of values that are used to calculate an average.

Returns

Double

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Decimal>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

```
C#
```

```
public static decimal Average (this System.Linq.ParallelQuery<decimal>
source);
```

Parameters

`source` ParallelQuery<Decimal>

A sequence of values that are used to calculate an average.

Returns

Decimal

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(ParallelQuery<Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values.

```
C#

public static decimal? Average (this System.Linq.ParallelQuery<decimal?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Decimal>>

A sequence of values that are used to calculate an average.

Returns

Nullable<Decimal>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static float Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static double? Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int64.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double? Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,int?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static double? Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static float? Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,float?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static double Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,int> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum or count of the elements in the sequence is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static double Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static decimal Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static decimal? Average<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal?>
```

```
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values that are used to calculate an average.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The average of the sequence of values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

OverflowException

(Thrown as inner exception in an AggregateException). The `selector` function returns a value greater than MaxValue for the element type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Cast<TResult> (ParallelQuery) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Converts the elements of a ParallelQuery to the specified type.

C#

```
public static System.Linq.ParallelQuery<TResult> Cast<TResult> (this
System.Linq.ParallelQuery source);
```

Type Parameters

TResult

The type to convert the elements of `source` to.

Parameters

source ParallelQuery

The sequence that contains the elements to be converted.

Returns

ParallelQuery<TResult>

A sequence that contains each element of the source sequence converted to the specified type.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

InvalidCastException

The type of the source sequence could not be converted to `TResult`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Concat Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Concatenates two parallel sequences.

Overloads

Expand table

Concat<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Concat<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Concatenates two parallel sequences.

Concat<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TSource> Concat<TSource> (this System.Linq.ParallelQuery<TSource> first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

TSource

This type parameter is not used.

Parameters

first ParallelQuery<TSource>

This parameter is not used.

second IEnumerable<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Concat with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Concat operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Concat<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Concatenates two parallel sequences.

C#

```
public static System.Linq.ParallelQuery<TSource> Concat<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

The first sequence to concatenate.

`second` ParallelQuery<TSource>

The sequence to concatenate to the first sequence.

Returns

ParallelQuery<TSource>

A sequence that contains the concatenated elements of the two input sequences.

Exceptions

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

ParallelEnumerable.Contains Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Determines in parallel whether a sequence contains a specified element.

Overloads

Expand table

Contains<TSource>(ParallelQuery<TSource>, TSource, IEqualityComparer<TSource>)	Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Contains<TSource>(ParallelQuery<TSource>, TSource)	Determines in parallel whether a sequence contains a specified element by using the default equality comparer.

Contains<TSource>(ParallelQuery<TSource>, TSource, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer<T>.

C#

```
public static bool Contains<TSource> (this
System.Linq.ParallelQuery<TSource> source, TSource value,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence in which to locate a value.

`value` TSource

The value to locate in the sequence.

`comparer` IEqualityComparer<TSource>

An equality comparer to compare values.

Returns

Boolean

`true` if the source sequence contains an element that has the specified value; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Contains<TSource>(ParallelQuery<TSource>, TSource)

Source: ParallelEnumerable.cs

Determines in parallel whether a sequence contains a specified element by using the default equality comparer.

C#

```
public static bool Contains<TSource> (this
System.Linq.ParallelQuery<TSource> source, TSource value);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence in which to locate a value.

`value` TSource

The value to locate in the sequence.

Returns

Boolean

`true` if the source sequence contains an element that has the specified value; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Count Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the number of elements in a parallel sequence.

Overloads

Expand table

Count<TSource>(ParallelQuery<TSource>)	Returns the number of elements in a parallel sequence.
Count<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.

Count<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the number of elements in a parallel sequence.

C#

```
public static int Count<TSource> (this System.Linq.ParallelQuery<TSource>
source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

A sequence that contains elements to be counted.

Returns

Int32

The number of elements in the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The number of elements in source is larger than Int32.MaxValue. (In this case the InnerException is OverflowException) -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Count<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.

```
C#

public static int Count<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence that contains elements to be counted.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Int32

A number that represents how many elements in the sequence satisfy the condition in the predicate function.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

The number of elements in source is larger than Int32.MaxValue. (In this case the InnerException is OverflowException) -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.DefaultIfEmpty Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.

Overloads

Expand table

DefaultIfEmpty<TSource> (ParallelQuery<TSource>)	Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource> (ParallelQuery<TSource>, TSource)	Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.

DefaultIfEmpty<TSource> (ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.

C#

```
public static System.Linq.ParallelQuery<TSource> DefaultIfEmpty<TSource>
(this System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return a default value for if it is empty.

Returns

ParallelQuery<TSource>

A sequence that contains default(TSource) if `source` is empty; otherwise, `source`.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

DefaultIfEmpty<TSource> (ParallelQuery<TSource>, TSource)

Source: ParallelEnumerable.cs

Returns the elements of the specified parallel sequence or the specified value in a singleton collection if the sequence is empty.

C#

```
public static System.Linq.ParallelQuery<TSource> DefaultIfEmpty<TSource>
(this System.Linq.ParallelQuery<TSource> source, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return the specified value for if it is empty.

`defaultValue` TSource

The value to return if the sequence is empty.

Returns

ParallelQuery<TSource>

A sequence that contains defaultValue if `source` is empty; otherwise, `source`.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Distinct Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.

Overloads

Expand table

Distinct<TSource>(ParallelQuery<TSource>)	Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.
Distinct<TSource>(ParallelQuery<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a parallel sequence by using a specified IEqualityComparer<T> to compare values.

Distinct<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.

C#

```
public static System.Linq.ParallelQuery<TSource> Distinct<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to remove duplicate elements from.

Returns

ParallelQuery<TSource>

A sequence that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Distinct<TSource>(ParallelQuery<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Returns distinct elements from a parallel sequence by using a specified IEqualityComparer<T> to compare values.

```
C#

public static System.Linq.ParallelQuery<TSource> Distinct<TSource> (this
System.Linq.ParallelQuery<TSource> source,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to remove duplicate elements from.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

ParallelQuery<TSource>

A sequence that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ElementAt<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Returns the element at a specified index in a parallel sequence.

C#

```
public static TSource ElementAt<TSource> (this
System.Linq.ParallelQuery<TSource> source, int index);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

A sequence to return an element from.

index Int32

The zero-based index of the element to retrieve.

Returns

TSource

The element at the specified position in the source sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

ArgumentOutOfRangeException

`index` is less than 0 or greater than or equal to the number of elements in `source`.

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ElementAtOrDefault<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Returns the element at a specified index in a parallel sequence or a default value if the index is out of range.

C#

```
public static TSource? ElementAtOrDefault<TSource> (this
System.Linq.ParallelQuery<TSource> source, int index);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

A sequence to return an element from.

index Int32

The zero-based index of the element to retrieve.

Returns

TSource

default(TSource) if the index is outside the bounds of the source sequence; otherwise, the element at the specified position in the source sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Empty<TResult> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Returns an empty ParallelQuery{TResult} that has the specified type argument.

C#

```
public static System.Linq.ParallelQuery<TResult> Empty<TResult> ();
```

Type Parameters

`TResult`

The type to assign to the type parameter of the returned generic sequence.

Returns

ParallelQuery<TResult>

An empty sequence whose type argument is `TResult`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Except Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Produces the set difference of two parallel sequences.

Overloads

Expand table

Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set difference of two parallel sequences by using the default equality comparer to compare values.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two parallel sequences by using the specified IEqualityComparer<T> to compare values.

Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TSource> Except<TSource> (this System.Linq.ParallelQuery<TSource> first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

TSource

This type parameter is not used.

Parameters

first ParallelQuery<TSource>

This parameter is not used.

second IEnumerable<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Except with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Except operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Produces the set difference of two parallel sequences by using the default equality comparer to compare values.

C#

```
public static System.Linq.ParallelQuery<TSource> Except<TSource> (this
System.Linq.ParallelQuery<TSource> first,
```

```
System.Linq.ParallelQuery<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence whose elements that are not also in `second` will be returned.

`second` ParallelQuery<TSource>

A sequence whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.

Returns

ParallelQuery<TSource>

A sequence that contains the set difference of the elements of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

> ⊗ **Caution**
>
> The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TSource> Except<TSource> (this System.Linq.ParallelQuery<TSource> first, System.Collections.Generic.IEnumerable<TSource> second, System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

This type parameter is not used.

Parameters

`first` ParallelQuery<TSource>

This parameter is not used.

`second` IEnumerable<TSource>

This parameter is not used.

`comparer` IEqualityComparer<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Except with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Except operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Produces the set difference of two parallel sequences by using the specified IEqualityComparer<T> to compare values.

C#

```
public static System.Linq.ParallelQuery<TSource> Except<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence whose elements that are not also in `second` will be returned.

`second` ParallelQuery<TSource>

A sequence whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.

`comparer` IEqualityComparer<TSource>

IEqualityComparer<T> to compare values.

Returns

ParallelQuery<TSource>

A sequence that contains the set difference of the elements of two sequences.

Exceptions

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.First Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the first element of a parallel sequence.

Overloads

Expand table

First<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence.
First<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element in a parallel sequence that satisfies a specified condition.

First<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the first element of a parallel sequence.

C#

```
public static TSource First<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

The sequence to return the first element of.

Returns

TSource

The first element in the specified sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

Remarks

If the query is not ordered, then the first element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

First<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns the first element in a parallel sequence that satisfies a specified condition.

```
C#

public static TSource First<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

The first element in the sequence that passes the test in the specified predicate function.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

No element in `source` satisfies the condition in `predicate`.

Remarks

If the query is not ordered, then the first element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ParallelEnumerable.FirstOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.

Overloads

Expand table

FirstOrDefault<TSource> (ParallelQuery<TSource>)	Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.

FirstOrDefault<TSource> (ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.

C#

```
public static TSource? FirstOrDefault<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return the first element of.

Returns

TSource

default(TSource) if `source` is empty; otherwise, the first element in `source`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

If the query is not ordered, then the first element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

FirstOrDefault<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.

C#

```
public static TSource? FirstOrDefault<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

default(TSource) if `source` is empty or if no element passes the test specified by predicate; otherwise, the first element in `source` that passes the test specified by predicate.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

If the query is not ordered, then the first element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.ForAll<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Invokes in parallel the specified action for each element in the `source`.

C#

```
public static void ForAll<TSource> (this System.Linq.ParallelQuery<TSource>
source, Action<TSource> action);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The ParallelQuery<TSource> whose elements will be processed by `action`.

`action` Action<TSource>

An Action to invoke on each element.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource> (ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

This is an efficient way to process the output from a parallelized query because it does not require a merge step at the end. For more information, see Merge Options in PLINQ or Order Preservation in PLINQ. Because this method produces a side-effect, it can be used to observe the actual order of execution in a parallel query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.GroupBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Groups in parallel the elements of a sequence according to a key selector function.

Overloads

Expand table

GroupBy<TSource,TKey,TElement,TResult> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified IComparer<T>.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function.

GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.

C#

```
public static System.Linq.ParallelQuery<TResult>
GroupBy<TSource,TKey,TElement,TResult> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TKey>
keySelector, Func<TSource,TElement> elementSelector,
Func<TKey,System.Collections.Generic.IEnumerable<TElement>,TResult>
```

```
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in each IGrouping{TKey, TElement}.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping{Key, TElement}.

`resultSelector` Func<TKey,IEnumerable<TElement>,TResult>

A function to create a result value from each group.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

ParallelQuery<TResult>

A sequence of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` or `keySelector` or `elementSelector` or `resultSelector` is `null` (`Nothing` in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>,

Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.

C#

```
public static System.Linq.ParallelQuery<TResult>
GroupBy<TSource,TKey,TElement,TResult> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TKey>
keySelector, Func<TSource,TElement> elementSelector,
Func<TKey,System.Collections.Generic.IEnumerable<TElement>,TResult>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in each IGrouping{TKey, TElement}.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping<TKey, TElement>.

`resultSelector` Func<TKey,IEnumerable<TElement>,TResult>

A function to create a result value from each group.

Returns

ParallelQuery<TResult>

A sequence of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` or `keySelector` or `elementSelector` or `resultSelector` is `null` (`Nothing` in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

GroupBy<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

C#

```
public static
System.Linq.ParallelQuery<System.Linq.IGrouping<TKey,TElement>>
GroupBy<TSource,TKey,TElement> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector, Func<TSource,TElement>
elementSelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in the IGrouping<TKey,TElement>.

Parameters

`source` ParallelQuery<TSource>

An OrderedParallelQuery<TSource> that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

Returns

ParallelQuery<IGrouping<TKey,TElement>>

A sequence of groups that are sorted descending according to `TKey`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

C#

```
public static
System.Linq.ParallelQuery<System.Linq.IGrouping<TKey,TElement>>
GroupBy<TSource,TKey,TElement> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector, Func<TSource,TElement>
elementSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the elements in the IGrouping.

Parameters

`source` ParallelQuery<TSource>

An OrderedParallelQuery{TSource}that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`elementSelector` Func<TSource,TElement>

A function to map each source element to an element in an IGrouping.

`comparer` IEqualityComparer<TKey>

An IComparer{TSource} to compare keys.

Returns

ParallelQuery<IGrouping<TKey,TElement>>

A sequence of groups that are sorted descending according to `TKey`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TResult> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

C#

```
public static System.Linq.ParallelQuery<TResult>
GroupBy<TSource,TKey,TResult> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector,
Func<TKey,System.Collections.Generic.IEnumerable<TSource>,TResult>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`resultSelector` Func<TKey,IEnumerable<TSource>,TResult>

A function to create a result value from each group.

Returns

ParallelQuery<TResult>

A sequence of elements of type `TResult` where each element represents a projection over a group and its key.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>,

Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.

```
C#

public static System.Linq.ParallelQuery<TResult>
GroupBy<TSource,TKey,TResult> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector,
Func<TKey,System.Collections.Generic.IEnumerable<TSource>,TResult>
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence whose elements to group.

`keySelector` Func<TSource,TKey>

A function to extract the key for each element.

`resultSelector` Func<TKey,IEnumerable<TSource>,TResult>

A function to create a result value from each group.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

ParallelQuery<TResult>

A sequence of groups.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified IComparer<T>.

```
C#

public static
System.Linq.ParallelQuery<System.Linq.IGrouping<TKey,TSource>>
GroupBy<TSource,TKey> (this System.Linq.ParallelQuery<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

An OrderedParallelQuery<TSource> that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`comparer` IEqualityComparer<TKey>

An IComparer<T> to compare keys.

Returns

ParallelQuery<IGrouping<TKey,TSource>>

A sequence of groups that are sorted descending according to `TKey`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Groups in parallel the elements of a sequence according to a specified key selector function.

C#

```
public static
System.Linq.ParallelQuery<System.Linq.IGrouping<TKey,TSource>>
```

```
GroupBy<TSource,TKey> (this System.Linq.ParallelQuery<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

An OrderedParallelQuery{TSource}that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

ParallelQuery<IGrouping<TKey,TSource>>

A sequence of groups that are sorted descending according to `TKey`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.GroupJoin Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Correlates in parallel the elements of two sequences based on equality of keys and groups the results.

Overloads

Expand table

GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.

GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TResult> GroupJoin<TOuter,TInner,TKey,TResult> (this System.Linq.ParallelQuery<TOuter> outer, System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey> outerKeySelector, Func<TInner,TKey> innerKeySelector, Func<TOuter,System.Collections.Generic.IEnumerable<TInner>,TResult> resultSelector);
```

Type Parameters

TOuter

This type parameter is not used.

TInner

This type parameter is not used.

`TKey`

This type parameter is not used.

`TResult`

This type parameter is not used.

Parameters

`outer` ParallelQuery<TOuter>

This parameter is not used.

`inner` IEnumerable<TInner>

This parameter is not used.

`outerKeySelector` Func<TOuter,TKey>

This parameter is not used.

`innerKeySelector` Func<TInner,TKey>

This parameter is not used.

`resultSelector` Func<TOuter,IEnumerable<TInner>,TResult>

This parameter is not used.

Returns

ParallelQuery<TResult>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of GroupJoin with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise,

the GroupJoin operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)

Source: ParallelEnumerable.cs

Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.

C#

```
public static System.Linq.ParallelQuery<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
System.Linq.ParallelQuery<TOuter> outer,
System.Linq.ParallelQuery<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
```

```
Func<TOuter,System.Collections.Generic.IEnumerable<TInner>,TResult>
resultSelector);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` ParallelQuery<TOuter>

The first sequence to join.

`inner` ParallelQuery<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,IEnumerable<TInner>,TResult>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

Returns

ParallelQuery<TResult>

A sequence that has elements of type `TResult` that are obtained by performing a grouped join on two sequences.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` or `outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null` (`Nothing` in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

⊗ **Caution**

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TResult> GroupJoin<TOuter,TInner,TKey,TResult> (this System.Linq.ParallelQuery<TOuter> outer, System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey> outerKeySelector, Func<TInner,TKey> innerKeySelector, Func<TOuter,System.Collections.Generic.IEnumerable<TInner>,TResult> resultSelector, System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TOuter`

This type parameter is not used.

`TInner`

This type parameter is not used.

`TKey`

This type parameter is not used.

`TResult`

This type parameter is not used.

Parameters

`outer` ParallelQuery<TOuter>

This parameter is not used.

`inner` IEnumerable<TInner>

This parameter is not used.

`outerKeySelector` Func<TOuter,TKey>

This parameter is not used.

`innerKeySelector` Func<TInner,TKey>

This parameter is not used.

`resultSelector` Func<TOuter,IEnumerable<TInner>,TResult>

This parameter is not used.

`comparer` IEqualityComparer<TKey>

This parameter is not used.

Returns

ParallelQuery<TResult>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of GroupJoin with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the GroupJoin operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

GroupJoin<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.

C#

```
public static System.Linq.ParallelQuery<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
System.Linq.ParallelQuery<TOuter> outer,
System.Linq.ParallelQuery<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
Func<TOuter,System.Collections.Generic.IEnumerable<TInner>,TResult>
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` ParallelQuery<TOuter>

The first sequence to join.

`inner` ParallelQuery<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,IEnumerable<TInner>,TResult>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to hash and compare keys.

Returns

ParallelQuery<TResult>

A sequence that has elements of type `TResult` that are obtained by performing a grouped join on two sequences.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` or `outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null` (`Nothing` in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Intersect Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Produces the set intersection of two parallel sequences.

Overloads

Expand table

Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two parallel sequences by using the specified IEqualityComparer{T} to compare values.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set intersection of two parallel sequences by using the default equality comparer to compare values.

Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

⊗ **Caution**

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TSource> Intersect<TSource> (this System.Linq.ParallelQuery<TSource> first, System.Collections.Generic.IEnumerable<TSource> second, System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

This type parameter is not used.

Parameters

`first` ParallelQuery<TSource>

This parameter is not used.

`second` IEnumerable<TSource>

This parameter is not used.

`comparer` IEqualityComparer<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Intersect with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Intersect operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Produces the set intersection of two parallel sequences by using the specified IEqualityComparer{T} to compare values.

```
C#

public static System.Linq.ParallelQuery<TSource> Intersect<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence whose distinct elements that also appear in `second` will be returned.

`second` ParallelQuery<TSource>

A sequence whose distinct elements that also appear in the first sequence will be returned.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

ParallelQuery<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)

Source: ParallelEnumerable.cs

> ⊗ **Caution**
>
> The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

```
C#
```

```
[System.Obsolete("The second data source of a binary operator must be of
type System.Linq.ParallelQuery<T> rather than
System.Collections.Generic.IEnumerable<T>. To fix this problem, use the
AsParallel() extension method to convert the right data source to
System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TSource> Intersect<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

`TSource`

This type parameter is not used.

Parameters

`first` ParallelQuery<TSource>

This parameter is not used.

`second` IEnumerable<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Intersect with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Intersect operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Produces the set intersection of two parallel sequences by using the default equality comparer to compare values.

C#

```
public static System.Linq.ParallelQuery<TSource> Intersect<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence whose distinct elements that also appear in `second` will be returned.

`second` ParallelQuery<TSource>

A sequence whose distinct elements that also appear in the first sequence will be returned.

Returns

ParallelQuery<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Join Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Correlates in parallel the elements of two sequences based on matching keys.

Overloads

Expand table

Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>,

Func<TInner,TKey>, Func<TOuter,TInner,TResult>)

Source: ParallelEnumerable.cs

> ⊗ **Caution**
>
> The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TResult> Join<TOuter,TInner,TKey,TResult> (this System.Linq.ParallelQuery<TOuter> outer, System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey> outerKeySelector, Func<TInner,TKey> innerKeySelector, Func<TOuter,TInner,TResult> resultSelector);
```

Type Parameters

`TOuter`

This type parameter is not used.

`TInner`

This type parameter is not used.

`TKey`

This type parameter is not used.

`TResult`

This type parameter is not used.

Parameters

`outer` ParallelQuery<TOuter>

This parameter is not used.

`inner` IEnumerable<TInner>

This parameter is not used.

`outerKeySelector` Func<TOuter,TKey>

This parameter is not used.

`innerKeySelector` Func<TInner,TKey>

This parameter is not used.

`resultSelector` Func<TOuter,TInner,TResult>

This parameter is not used.

Returns

ParallelQuery<TResult>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage Join with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Join operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)

Source: ParallelEnumerable.cs

Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.

C#

```
public static System.Linq.ParallelQuery<TResult>
Join<TOuter,TInner,TKey,TResult> (this System.Linq.ParallelQuery<TOuter>
outer, System.Linq.ParallelQuery<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
Func<TOuter,TInner,TResult> resultSelector);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` ParallelQuery<TOuter>

The first sequence to join.

`inner` ParallelQuery<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,TInner,TResult>

A function to create a result element from two matching elements.

Returns

ParallelQuery<TResult>

A sequence that has elements of type `TResult` that are obtained by performing an inner join on two sequences.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` or `outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null` (`Nothing` in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TResult> Join<TOuter,TInner,TKey,TResult> (this System.Linq.ParallelQuery<TOuter> outer, System.Collections.Generic.IEnumerable<TInner> inner, Func<TOuter,TKey> outerKeySelector, Func<TInner,TKey> innerKeySelector, Func<TOuter,TInner,TResult> resultSelector, System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TOuter`

This type parameter is not used.

`TInner`

This type parameter is not used.

`TKey`

This type parameter is not used.

`TResult`

This type parameter is not used.

Parameters

`outer` ParallelQuery<TOuter>

This parameter is not used.

`inner` IEnumerable<TInner>

This parameter is not used.

`outerKeySelector` Func<TOuter,TKey>

This parameter is not used.

`innerKeySelector` Func<TInner,TKey>

This parameter is not used.

`resultSelector` Func<TOuter,TInner,TResult>

This parameter is not used.

`comparer` IEqualityComparer<TKey>

This parameter is not used.

Returns

ParallelQuery<TResult>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Join with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Join operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*

Product	Versions *(Obsolete)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Join<TOuter,TInner,TKey,TResult> (ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

C#

```
public static System.Linq.ParallelQuery<TResult>
Join<TOuter,TInner,TKey,TResult> (this System.Linq.ParallelQuery<TOuter>
outer, System.Linq.ParallelQuery<TInner> inner, Func<TOuter,TKey>
outerKeySelector, Func<TInner,TKey> innerKeySelector,
Func<TOuter,TInner,TResult> resultSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` ParallelQuery<TOuter>

The first sequence to join.

`inner` ParallelQuery<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Func<TOuter,TKey>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Func<TInner,TKey>

A function to extract the join key from each element of the second sequence.

`resultSelector` Func<TOuter,TInner,TResult>

A function to create a result element from two matching elements.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to hash and compare keys.

Returns

ParallelQuery<TResult>

A sequence that has elements of type `TResult` that are obtained by performing an inner join on two sequences.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `action` or `outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null` (`Nothing` in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Last Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the last element of a parallel sequence.

Overloads

Expand table

Last<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence.
Last<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a specified condition.

Last<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the last element of a parallel sequence.

C#

```
public static TSource Last<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

The sequence to return the last element from.

Returns

TSource

The value at the last position in the source sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

Remarks

If the query is not ordered, then the last element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Last<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns the last element of a parallel sequence that satisfies a specified condition.

```
C#

public static TSource Last<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

The last element in the sequence that passes the test in the specified predicate function.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

No element in `source` satisfies the condition in `predicate`.

Remarks

If the query is not ordered, then the last element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ParallelEnumerable.LastOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.

Overloads

Expand table

LastOrDefault<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.
LastOrDefault<TSource> (ParallelQuery<TSource>)	Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.

LastOrDefault<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.

C#

```
public static TSource? LastOrDefault<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return an element from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

TSource

default() if the sequence is empty or if no elements pass the test in the predicate function; otherwise, the last element that passes the test in the predicate function.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

If the query is not ordered, then the last element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

LastOrDefault<TSource> (ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.

C#

```
public static TSource? LastOrDefault<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return an element from.

Returns

TSource

default() if the source sequence is empty; otherwise, the last element in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

If the query is not ordered, then the last element is non-deterministic. For more information, see Order Preservation in PLINQ.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.LongCount Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns an Int64 that represents the total number of elements in a parallel sequence.

Overloads

Expand table

LongCount<TSource> (ParallelQuery<TSource>)	Returns an Int64 that represents the total number of elements in a parallel sequence.
LongCount<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.

LongCount<TSource> (ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns an Int64 that represents the total number of elements in a parallel sequence.

C#

```
public static long LongCount<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence that contains elements to be counted.

Returns

Int64

The number of elements in the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The number of elements in source is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

OverflowException

The computed result is greater than Int32.MaxValue.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

LongCount<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.

```
C#

public static long LongCount<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence that contains elements to be counted.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

Int64

A number that represents how many elements in the sequence satisfy the condition in the predicate function.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

The number of elements in source is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

OverflowException

The computed result is greater than Int32.MaxValue.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Max Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the maximum value in a parallel sequence of values.

Overloads

Expand table

Max(ParallelQuery<Nullable<Single>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Single>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Int64>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Int32>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Double>>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Double>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Int64>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Int32>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Decimal>)	Returns the maximum value in a parallel sequence of values.
Max(ParallelQuery<Nullable<Decimal>>)	Returns the maximum value in a parallel sequence of values.

Max<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>)	Returns the maximum value in a parallel sequence of values.

Max(ParallelQuery<Nullable<Single>>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

```C#
public static float? Max (this System.Linq.ParallelQuery<float?> source);
```

Parameters

`source` ParallelQuery<Nullable<Single>>

A sequence of values to determine the maximum value of.

Returns

Nullable<Single>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Single>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

C#

```
public static float Max (this System.Linq.ParallelQuery<float> source);
```

Parameters

`source` ParallelQuery<Single>

A sequence of values to determine the maximum value of.

Returns

Single

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Nullable<Int64>>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

C#

```
public static long? Max (this System.Linq.ParallelQuery<long?> source);
```

Parameters

`source` ParallelQuery<Nullable<Int64>>

A sequence of values to determine the maximum value of.

Returns

Nullable<Int64>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Nullable<Int32>>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

```
C#
public static int? Max (this System.Linq.ParallelQuery<int?> source);
```

Parameters

`source` ParallelQuery<Nullable<Int32>>

A sequence of values to determine the maximum value of.

Returns

Nullable<Int32>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Nullable<Double>>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

```
C#

public static double? Max (this System.Linq.ParallelQuery<double?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Double>>

A sequence of values to determine the maximum value of.

Returns

Nullable<Double>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Double>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

C#

```
public static double Max (this System.Linq.ParallelQuery<double> source);
```

Parameters

`source` ParallelQuery<Double>

A sequence of values to determine the maximum value of.

Returns

Double

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Int64>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

```
C#
```

```
public static long Max (this System.Linq.ParallelQuery<long> source);
```

Parameters

`source` ParallelQuery<Int64>

A sequence of values to determine the maximum value of.

Returns

Int64

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Int32>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

C#

```
public static int Max (this System.Linq.ParallelQuery<int> source);
```

Parameters

`source` ParallelQuery<Int32>

A sequence of values to determine the maximum value of.

Returns

Int32

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Decimal>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

C#

```
public static decimal Max (this System.Linq.ParallelQuery<decimal>
source);
```

Parameters

`source` ParallelQuery<Decimal>

A sequence of values to determine the maximum value of.

Returns

Decimal

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max(ParallelQuery<Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

```
C#
```

```
public static decimal? Max (this System.Linq.ParallelQuery<decimal?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Decimal>>

A sequence of values to determine the maximum value of.

Returns

Nullable<Decimal>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,TResult>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static TResult? Max<TSource,TResult> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TResult>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TResult`

The type of the value returned by `selector`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,TResult>

A transform function to apply to each element.

Returns

TResult

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

```
C#

public static long? Max<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Int64>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static float Max<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

```
C#

public static float? Max<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,float?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static int? Max<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Int32>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static double? Max<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

```C#
public static decimal? Max<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static long Max<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Int64

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

```
C#

public static int Max<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Int32

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static double Max<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the maximum value.

C#

```
public static decimal Max<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the maximum value in a parallel sequence of values.

```
C#
```

```
public static TSource? Max<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of elements in the source sequence.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the maximum value of.

Returns

TSource

The maximum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Min Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the minimum value in a parallel sequence of values.

Overloads

Expand table

Min(ParallelQuery<Single>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Single>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Int64>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Int32>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Double>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Nullable<Decimal>>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Int64>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Int32>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Double>)	Returns the minimum value in a parallel sequence of values.
Min(ParallelQuery<Decimal>)	Returns the minimum value in a parallel sequence of values.

Min<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>)	Returns the minimum value in a parallel sequence of values.

Min(ParallelQuery<Single>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

```
C#

public static float Min (this System.Linq.ParallelQuery<float> source);
```

Parameters

`source` ParallelQuery<Single>

A sequence of values to determine the minimum value of.

Returns

Single

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Nullable<Single>>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

C#

```
public static float? Min (this System.Linq.ParallelQuery<float?> source);
```

Parameters

`source` ParallelQuery<Nullable<Single>>

A sequence of values to determine the minimum value of.

Returns

Nullable<Single>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Nullable<Int64>>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

C#

```
public static long? Min (this System.Linq.ParallelQuery<long?> source);
```

Parameters

`source` ParallelQuery<Nullable<Int64>>

A sequence of values to determine the minimum value of.

Returns

Nullable<Int64>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Nullable<Int32>>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

```C#
public static int? Min (this System.Linq.ParallelQuery<int?> source);
```

Parameters

`source` ParallelQuery<Nullable<Int32>>

A sequence of values to determine the minimum value of.

Returns

Nullable<Int32>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Nullable<Double>>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

```
C#

public static double? Min (this System.Linq.ParallelQuery<double?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Double>>

A sequence of values to determine the minimum value of.

Returns

Nullable<Double>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

C#

```
public static decimal? Min (this System.Linq.ParallelQuery<decimal?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Decimal>>

A sequence of values to determine the minimum value of.

Returns

Nullable<Decimal>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Int64>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

```
C#
public static long Min (this System.Linq.ParallelQuery<long> source);
```

Parameters

`source` ParallelQuery<Int64>

A sequence of values to determine the minimum value of.

Returns

Int64

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Int32>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

C#

```
public static int Min (this System.Linq.ParallelQuery<int> source);
```

Parameters

`source` ParallelQuery<Int32>

A sequence of values to determine the minimum value of.

Returns

Int32

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Double>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

C#

```
public static double Min (this System.Linq.ParallelQuery<double> source);
```

Parameters

`source` ParallelQuery<Double>

A sequence of values to determine the minimum value of.

Returns

Double

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min(ParallelQuery<Decimal>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

```
C#
```

```
public static decimal Min (this System.Linq.ParallelQuery<decimal>
source);
```

Parameters

`source` ParallelQuery<Decimal>

A sequence of values to determine the minimum value of.

Returns

Decimal

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,TResult>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

C#

```
public static TResult? Min<TSource,TResult> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TResult>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TResult`

The type of the value returned by `selector`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,TResult>

A transform function to apply to each element.

Returns

TResult

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

```
C#

public static float Min<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

C#

```
public static float? Min<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,float?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

```
C#

public static int? Min<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Int32>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

```
C#
```

```
public static double? Min<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

C#

```
public static long? Min<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Int64>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

```
C#

public static long Min<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Int64

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

C#

```
public static int Min<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Int32

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

C#

```
public static decimal? Min<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

C#

```
public static double Min<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)

Source: ParallelEnumerable.cs

Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

```
C#

public static decimal Min<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the minimum value in a parallel sequence of values.

```
C#

public static TSource? Min<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to determine the minimum value of.

Returns

TSource

The minimum value in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` contains no elements and `TSource` is a non-nullable value type.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.OfType<TResult> (ParallelQuery) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Filters the elements of a ParallelQuery based on a specified type.

C#

```
public static System.Linq.ParallelQuery<TResult> OfType<TResult> (this
System.Linq.ParallelQuery source);
```

Type Parameters

`TResult`

The type to filter the elements of the sequence on.

Parameters

`source` ParallelQuery

The sequence with elements to filter.

Returns

ParallelQuery<TResult>

A sequence that contains elements from the input sequence filtered by type.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.OrderBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Sorts in parallel the elements of a sequence in ascending order.

Overloads

Expand table

OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.

OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Sorts in parallel the elements of a sequence in ascending order according to a key.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
OrderBy<TSource,TKey> (this System.Linq.ParallelQuery<TSource> source,
Func<TSource,TKey> keySelector);
```

Type Parameters

TSource

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. To achieve a stable sort, change a query of the form: var ordered = source.Select((e,i) => new { E=e, I=i }).OrderBy((v) => v.e).ThenBy(v => v.i).Select((v) => v.e);

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

OrderBy<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: ParallelEnumerable.cs

Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
OrderBy<TSource,TKey> (this System.Linq.ParallelQuery<TSource> source,
Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer{TKey} to compare keys.

Returns

OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.OrderByDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Sorts the elements of a sequence in descending order.

Overloads

Expand table

OrderByDescending<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.

OrderByDescending<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Sorts in parallel the elements of a sequence in descending order according to a key.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
OrderByDescending<TSource,TKey> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

OrderByDescending<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: ParallelEnumerable.cs

Sorts the elements of a sequence in descending order by using a specified comparer.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
OrderByDescending<TSource,TKey> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to order.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer{TKey} to compare keys.

Returns

OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} whose elements are sorted descending according to a key.

Exceptions

ArgumentNullException

`source` or `KeySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Range(Int32, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Generates a parallel sequence of integral numbers within a specified range.

C#

```
public static System.Linq.ParallelQuery<int> Range (int start, int count);
```

Parameters

`start` Int32

The value of the first integer in the sequence.

`count` Int32

The number of sequential integers to generate.

Returns

ParallelQuery<Int32>

An IEnumerable<Int32> in C# or IEnumerable(Of Int32) in Visual Basic that contains a range of sequential integral numbers.

Exceptions

ArgumentOutOfRangeException

`count` is less than 0 -or- `start` + `count` - 1 is larger than Int32.MaxValue.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Repeat<TResult> (TResult, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Generates a parallel sequence that contains one repeated value.

C#

```
public static System.Linq.ParallelQuery<TResult> Repeat<TResult> (TResult
element, int count);
```

Type Parameters

TResult

The type of the value to be repeated in the result sequence.

Parameters

element TResult

The value to be repeated.

count Int32

The number of times to repeat the value in the generated sequence.

Returns

ParallelQuery<TResult>

A sequence that contains a repeated value.

Exceptions

ArgumentOutOfRangeException

`count` is less than 0.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Reverse<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Inverts the order of the elements in a parallel sequence.

C#

```
public static System.Linq.ParallelQuery<TSource> Reverse<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to reverse.

Returns

ParallelQuery<TSource>

A sequence whose elements correspond to those of the input sequence in reverse order.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Select Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Projects in parallel each element of a sequence into a new form.

Overloads

Expand table

Select<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,Int32,TResult>)	Projects in parallel each element of a sequence into a new form by incorporating the element's index.
Select<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,TResult>)	Projects in parallel each element of a sequence into a new form.

Select<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,Int32,TResult>)

Source: ParallelEnumerable.cs

Projects in parallel each element of a sequence into a new form by incorporating the element's index.

C#

```
public static System.Linq.ParallelQuery<TResult> Select<TSource,TResult>
(this System.Linq.ParallelQuery<TSource> source,
Func<TSource,int,TResult> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of elements returned by selector.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to invoke a transform function on.

`selector` Func<TSource,Int32,TResult>

A transform function to apply to each element.

Returns

ParallelQuery<TResult>

A sequence whose elements are the result of invoking the transform function on each element of `source`, based on the index supplied to `selector`.

Exceptions

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

OverflowException

More than Int32.MaxValue elements are enumerated by the query. This condition might occur in streaming scenarios.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Select<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,TResult>)

Source: ParallelEnumerable.cs

Projects in parallel each element of a sequence into a new form.

C#

```
public static System.Linq.ParallelQuery<TResult> Select<TSource,TResult>
(this System.Linq.ParallelQuery<TSource> source, Func<TSource,TResult>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of elements returned by selector.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to invoke a transform function on.

`selector` Func<TSource,TResult>

A transform function to apply to each element.

Returns

ParallelQuery<TResult>

A sequence whose elements are the result of invoking the transform function on each element of `source`.

Exceptions

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.SelectMany Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Projects in parallel each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.

Overloads

Expand table

SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)

Source: ParallelEnumerable.cs

Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.

C#

```
public static System.Linq.ParallelQuery<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Linq.ParallelQuery<TSource> source,
Func<TSource,System.Collections.Generic.IEnumerable<TCollection>>
collectionSelector, Func<TSource,TCollection,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the intermediate elements collected by `collectionSelector`.

`TCollection`

The type of elements of `source`.

`TResult`

The type of elements in the result sequence.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to project.

`collectionSelector` Func<TSource,IEnumerable<TCollection>>

A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

`resultSelector` Func<TSource,TCollection,TResult>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

Returns

ParallelQuery<TResult>

A sequence whose elements are the result of invoking the one-to-many transform function `collectionSelector` on each element of `source` based on the index supplied to `collectionSelector`, and then mapping each of those sequence elements and their corresponding source element to a result element.

Exceptions

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

OverflowException

More than Int32.MaxValue elements are enumerated by the query.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)

Source: ParallelEnumerable.cs

Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.

C#

```
public static System.Linq.ParallelQuery<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Linq.ParallelQuery<TSource> source,
Func<TSource,int,System.Collections.Generic.IEnumerable<TCollection>>
collectionSelector, Func<TSource,TCollection,TResult> resultSelector);
```

Type Parameters

`TSource`

The type of the intermediate elements collected by `collectionSelector`.

`TCollection`

The type of elements of `source`.

`TResult`

The type of elements to return.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to project.

`collectionSelector` Func<TSource,Int32,IEnumerable<TCollection>>

A transform function to apply to each source element; the second parameter of the function represents the index of the source element.

`resultSelector` Func<TSource,TCollection,TResult>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

Returns

ParallelQuery<TResult>

A sequence whose elements are the result of invoking the one-to-many transform function `collectionSelector` on each element of `source` based on the index supplied to `collectionSelector`, and then mapping each of those sequence elements and their corresponding source element to a result element.

Exceptions

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

OverflowException

More than Int32.MaxValue elements are enumerated by the query.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SelectMany<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,IEnumerable<TResult>>)

Source: ParallelEnumerable.cs

Projects in parallel each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.

C#

```
public static System.Linq.ParallelQuery<TResult>
SelectMany<TSource,TResult> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,System.Collections.Generic.IEnumerable<TResult>>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TResult`

The type of the elements of the sequence returned by selector.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to project.

`selector` Func<TSource,IEnumerable<TResult>>

A transform function to apply to each element.

Returns

ParallelQuery<TResult>

A sequence whose elements are the result of invoking the one-to-many transform function on each element of the input sequence.

Exceptions

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SelectMany<TSource,TResult> (ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)

Source: ParallelEnumerable.cs

Projects in parallel each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

```
C#

public static System.Linq.ParallelQuery<TResult>
SelectMany<TSource,TResult> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int,System.Collections.Generic.IEnumerable<TResult>>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TResult`

The type of the elements of the sequence returned by selector.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to project.

`selector` Func<TSource,Int32,IEnumerable<TResult>>

A transform function to apply to each element.

Returns

ParallelQuery<TResult>

A sequence whose elements are the result of invoking the one-to-many transform function on each element of the input sequence.

Exceptions

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

OverflowException

More than Int32.MaxValue elements are enumerated by the query.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.

ParallelEnumerable.SequenceEqual Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Determines whether two parallel sequences are equal by comparing the elements by using an equality comparer for their type.

Overloads

Expand table

SequenceEqual<TSource> (ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
SequenceEqual<TSource> (ParallelQuery<TSource>, ParallelQuery<TSource>)	Determines whether two parallel sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource> (ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
SequenceEqual<TSource> (ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Determines whether two parallel sequences are equal by comparing their elements by using a specified IEqualityComparer{T}.

SequenceEqual<TSource> (ParallelQuery<TSource>,

IEnumerable<TSource>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static bool SequenceEqual<TSource> (this System.Linq.ParallelQuery<TSource> first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

TSource

This type parameter is not used.

Parameters

first ParallelQuery<TSource>

This parameter is not used.

second IEnumerable<TSource>

This parameter is not used.

Returns

Boolean

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Thrown every time this method is called.

Remarks

This overload exists to disallow usage of SequenceEqual with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the SequenceEqual operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

SequenceEqual<TSource> (ParallelQuery<TSource>, ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Determines whether two parallel sequences are equal by comparing the elements by using the default equality comparer for their type.

```
C#

public static bool SequenceEqual<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence to compare to second.

`second` ParallelQuery<TSource>

A sequence to compare to the first input sequence.

Returns

Boolean

`true` if the two source sequences are of equal length and their corresponding elements are equal according to the default equality comparer for their type; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SequenceEqual<TSource> (ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of
type System.Linq.ParallelQuery<T> rather than
System.Collections.Generic.IEnumerable<T>. To fix this problem, use the
AsParallel() extension method to convert the right data source to
System.Linq.ParallelQuery<T>.")]
public static bool SequenceEqual<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

This type parameter is not used.

Parameters

`first` ParallelQuery<TSource>

This parameter is not used.

`second` IEnumerable<TSource>

This parameter is not used.

`comparer` IEqualityComparer<TSource>

This parameter is not used.

Returns

Boolean

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

Thrown every time this method is called.

Remarks

This overload exists to disallow usage of SequenceEqual with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the SequenceEqual operator would appear to be binding to the parallel implementation, but would in reality bind to sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

SequenceEqual<TSource> (ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Determines whether two parallel sequences are equal by comparing their elements by using a specified IEqualityComparer{T}.

C#

```
public static bool SequenceEqual<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence to compare to `second`.

`second` ParallelQuery<TSource>

A sequence to compare to the first input sequence.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to use to compare elements.

Returns

Boolean

`true` if the two source sequences are of equal length and their corresponding elements are equal according to the default equality comparer for their type; otherwise, `false`.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Single Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.

Overloads

Expand table

Single<TSource> (ParallelQuery<TSource>)	Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

Single<TSource>(ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.

C#

```
public static TSource Single<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return the single element of.

Returns

TSource

The single element of the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

The input sequence contains more than one element. -or- The input sequence is empty.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Single<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns the only element of a parallel sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

C#

```
public static TSource Single<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return the single element of.

`predicate` Func<TSource,Boolean>

A function to test an element for a condition.

Returns

TSource

The single element of the input sequence that satisfies a condition.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

No element satisfies the condition in `predicate`. -or- More than one element satisfies the condition in `predicate`.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.SingleOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

Overloads

Expand table

SingleOrDefault<TSource> (ParallelQuery<TSource>)	Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

SingleOrDefault<TSource> (ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

C#

```
public static TSource? SingleOrDefault<TSource> (this
```

```
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return the single element of.

Returns

TSource

The single element of the input sequence, or default() if the sequence contains no elements.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

InvalidOperationException

`source` has more than one element.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SingleOrDefault<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

C#

```
public static TSource? SingleOrDefault<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return the single element of.

`predicate` Func<TSource,Boolean>

A function to test an element for a condition.

Returns

TSource

The single element of the input sequence that satisfies the condition, or default() if no such element is found.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

InvalidOperationException

`source` is empty or multiple elements are returned.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Skip<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Bypasses a specified number of elements in a parallel sequence and then returns the remaining elements.

C#

```
public static System.Linq.ParallelQuery<TSource> Skip<TSource> (this
System.Linq.ParallelQuery<TSource> source, int count);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return elements from.

`count` Int32

The number of elements to skip before returning the remaining elements.

Returns

ParallelQuery<TSource>

A sequence that contains the elements that occur after the specified index in the input sequence.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource> (ParallelQuery<TSource>, CancellationToken).

OverflowException

Count is greater than Int32.MaxValue

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

If count > source.Count(), no elements are returned.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

ParallelEnumerable.SkipWhile Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.

Overloads

Expand table

SkipWhile<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource> (ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.

C#

```
public static System.Linq.ParallelQuery<TSource> SkipWhile<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return elements from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

ParallelQuery<TSource>

A sequence that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by predicate.

Exceptions

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)

Source: ParallelEnumerable.cs

Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

C#

```
public static System.Linq.ParallelQuery<TSource> SkipWhile<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,int,bool>
predicate);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return elements from.

`predicate` Func<TSource,Int32,Boolean>

A function to test each source element for a condition; the second parameter of the function represents the index of the source element.

Returns

ParallelQuery<TSource>

A sequence that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by predicate.

Exceptions

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

OverflowException

More than Int32.MaxValue elements are enumerated by the query.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

.NET feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Sum Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Computes in parallel the sum of a sequence of values.

Overloads

Expand table

Sum(ParallelQuery<Nullable<Single>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Int64>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Int32>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Double>>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Single>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Int64>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Int32>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Double>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Decimal>)	Computes in parallel the sum of a sequence of values.
Sum(ParallelQuery<Nullable<Decimal>>)	Computes in parallel the sum of a sequence of values.

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

Sum(ParallelQuery<Nullable<Single>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

```
C#

public static float? Sum (this System.Linq.ParallelQuery<float?> source);
```

Parameters

`source` ParallelQuery<Nullable<Single>>

A sequence of values to calculate the sum of.

Returns

Nullable<Single>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Single.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Nullable<Int64>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static long? Sum (this System.Linq.ParallelQuery<long?> source);
```

Parameters

`source` ParallelQuery<Nullable<Int64>>

A sequence of values to calculate the sum of.

Returns

Nullable<Int64>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int64.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Nullable<Int32>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static int? Sum (this System.Linq.ParallelQuery<int?> source);
```

Parameters

`source` ParallelQuery<Nullable<Int32>>

A sequence of values to calculate the sum of.

Returns

Nullable<Int32>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Nullable<Double>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static double? Sum (this System.Linq.ParallelQuery<double?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Double>>

A sequence of values to calculate the sum of.

Returns

Nullable<Double>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Double.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Single>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

```
C#
public static float Sum (this System.Linq.ParallelQuery<float> source);
```

Parameters

`source` ParallelQuery<Single>

A sequence of values to calculate the sum of.

Returns

Single

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Single.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Int64>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static long Sum (this System.Linq.ParallelQuery<long> source);
```

Parameters

`source` ParallelQuery<Int64>

A sequence of values to calculate the sum of.

Returns

Int64

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int64.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Int32>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static int Sum (this System.Linq.ParallelQuery<int> source);
```

Parameters

`source` ParallelQuery<Int32>

A sequence of values to calculate the sum of.

Returns

Int32

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Double>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static double Sum (this System.Linq.ParallelQuery<double> source);
```

Parameters

`source` ParallelQuery<Double>

A sequence of values to calculate the sum of.

Returns

Double

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Double.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Decimal>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

```
C#

public static decimal Sum (this System.Linq.ParallelQuery<decimal>
source);
```

Parameters

`source` ParallelQuery<Decimal>

A sequence of values to calculate the sum of.

Returns

Decimal

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Decimal.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(ParallelQuery<Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of a sequence of values.

C#

```
public static decimal? Sum (this System.Linq.ParallelQuery<decimal?>
source);
```

Parameters

`source` ParallelQuery<Nullable<Decimal>>

A sequence of values to calculate the sum of.

Returns

Nullable<Decimal>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Decimal.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static float Sum<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,float> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Single>

A transform function to apply to each element.

Returns

Single

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Single.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static decimal Sum<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Decimal>

A transform function to apply to each element.

Returns

Decimal

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Decimal.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static double Sum<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Double>

A transform function to apply to each element.

Returns

Double

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Double.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static int Sum<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int> selector);
```

Type Parameters

`TSource`

The type of elements in the source sequence.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Int32>

A transform function to apply to each element.

Returns

Int32

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static long Sum<TSource> (this System.Linq.ParallelQuery<TSource>
```

```
source, Func<TSource,long> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Int64>

A transform function to apply to each element.

Returns

Int64

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int64.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static decimal? Sum<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,decimal?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Nullable<Decimal>>

A transform function to apply to each element.

Returns

Nullable<Decimal>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Decimal.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static double? Sum<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,double?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Nullable<Double>>

A transform function to apply to each element.

Returns

Nullable<Double>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Double.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static int? Sum<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,int?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Nullable<Int32>>

A transform function to apply to each element.

Returns

Nullable<Int32>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int32.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

```
C#

public static long? Sum<TSource> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,long?> selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Nullable<Int64>>

A transform function to apply to each element.

Returns

Nullable<Int64>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Int64.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)

Source: ParallelEnumerable.cs

Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.

C#

```
public static float? Sum<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,float?>
selector);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A sequence of values to calculate the sum of.

`selector` Func<TSource,Nullable<Single>>

A transform function to apply to each element.

Returns

Nullable<Single>

The sum of the projected values in the sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `selector` is a null reference (Nothing in Visual Basic).

AggregateException

The sum is larger than Single.MaxValue. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Take<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Returns a specified number of contiguous elements from the start of a parallel sequence.

C#

```
public static System.Linq.ParallelQuery<TSource> Take<TSource> (this
System.Linq.ParallelQuery<TSource> source, int count);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return elements from.

`count` Int32

The number of elements to return.

Returns

ParallelQuery<TSource>

A sequence that contains the specified number of elements from the start of the input sequence.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource> (ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.TakeWhile Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Returns elements from a parallel sequence as long as a specified condition is true.

Overloads

Expand table

TakeWhile<TSource> (ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a parallel sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
TakeWhile<TSource> (ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns elements from a parallel sequence as long as a specified condition is true.

TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)

Source: ParallelEnumerable.cs

Returns elements from a parallel sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

C#

```
public static System.Linq.ParallelQuery<TSource> TakeWhile<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,int,bool>
predicate);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return elements from.

`predicate` Func<TSource,Int32,Boolean>

A function to test each source element for a condition; the second parameter of the function represents the index of the source element.

Returns

ParallelQuery<TSource>

A sequence that contains elements from the input sequence that occur before the element at which the test no longer passes.

Exceptions

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

OverflowException

More than Int32.MaxValue elements are enumerated by this query.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Returns elements from a parallel sequence as long as a specified condition is true.

C#

```
public static System.Linq.ParallelQuery<TSource> TakeWhile<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

The sequence to return elements from.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

ParallelQuery<TSource>

A sequence that contains the elements from the input sequence that occur before the element at which the test no longer passes.

Exceptions

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ThenBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Performs in parallel a subsequent ordering of the elements in a sequence in ascending order.

Overloads

Expand table

ThenBy<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

ThenBy<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Performs in parallel a subsequent ordering of the elements in a sequence in ascending order according to a key.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
ThenBy<TSource,TKey> (this System.Linq.OrderedParallelQuery<TSource>
source, Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ThenBy<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: ParallelEnumerable.cs

Performs in parallel a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
ThenBy<TSource,TKey> (this System.Linq.OrderedParallelQuery<TSource>
source, Func<TSource,TKey> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer{TKey} to compare keys.

Returns

OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.ThenByDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Performs in parallel a subsequent ordering of the elements in a sequence in descending order.

Overloads

Expand table

ThenByDescending<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Performs in parallel a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

ThenByDescending<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Performs in parallel a subsequent ordering of the elements in a sequence in descending order, according to a key.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
ThenByDescending<TSource,TKey> (this
```

```
System.Linq.OrderedParallelQuery<TSource> source, Func<TSource,TKey>
keySelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

Returns

OrderedParallelQuery<TSource>

A sequence whose elements are sorted descending according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ThenByDescending<TSource,TKey> (OrderedParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)

Source: ParallelEnumerable.cs

Performs in parallel a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

C#

```
public static System.Linq.OrderedParallelQuery<TSource>
ThenByDescending<TSource,TKey> (this
System.Linq.OrderedParallelQuery<TSource> source, Func<TSource,TKey>
keySelector, System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` OrderedParallelQuery<TSource>

An OrderedParallelQuery{TSource} that contains elements to sort.

`keySelector` Func<TSource,TKey>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer{TKey} to compare keys.

Returns

OrderedParallelQuery<TSource>

A sequence whose elements are sorted descending according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

In contrast to the sequential implementation, this is not a stable sort. See the remarks for OrderBy for an approach to implementing a stable sort.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ToArray<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Creates an array from a ParallelQuery<TSource>.

C#

```
public static TSource[] ToArray<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

A sequence to create an array from.

Returns

TSource[]

An array that contains the elements from the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource> (ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ToDictionary Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource>.

Overloads

Expand table

ToDictionary<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function, a comparer, and an element selector function.
ToDictionary<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function.
ToDictionary<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.

ToDictionary<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)

Source: ParallelEnumerable.cs

Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to specified key selector and element selector functions.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TSource,TKey,TElement> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TKey>
keySelector, Func<TSource,TElement> elementSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

Returns

Dictionary<TKey,TElement>

A Dictionary<TKey,TValue> that contains values of type `TElement` selected from the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` or `elementSelector` is a null reference (Nothing in Visual Basic).

AggregateException

`keySelector` produces a key that is a null reference (Nothing in Visual Basic). -or- `keySelector` produces duplicate keys for two elements. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function, a comparer, and an element selector function.

```
C#

public static System.Collections.Generic.Dictionary<TKey,TElement>
ToDictionary<TSource,TKey,TElement> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,TKey>
keySelector, Func<TSource,TElement> elementSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

Dictionary<TKey,TElement>

A Dictionary<TKey,TValue> that contains values of type `TElement` selected from the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` or `elementSelector` is a null reference (Nothing in Visual Basic).

AggregateException

`keySelector` produces a key that is a null reference (Nothing in Visual Basic). -or- `keySelector` produces duplicate keys for two elements. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TSource>
ToDictionary<TSource,TKey> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

Returns

Dictionary<TKey,TSource>

A Dictionary<TKey,TValue> that contains keys and values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

AggregateException

`keySelector` produces a key that is a null reference (Nothing in Visual Basic). -or- `keySelector` produces duplicate keys for two elements. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.

C#

```
public static System.Collections.Generic.Dictionary<TKey,TSource>
ToDictionary<TSource,TKey> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

A sequence to create a Dictionary<TKey,TValue> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

Dictionary<TKey,TSource>

A Dictionary<TKey,TValue> that contains keys and values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

AggregateException

`keySelector` produces a key that is a null reference (Nothing in Visual Basic). -or- `keySelector` produces duplicate keys for two elements. -or- One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ToList<TSource> (ParallelQuery<TSource>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Creates a List<T> from an ParallelQuery<TSource>.

C#

```
public static System.Collections.Generic.List<TSource> ToList<TSource> (this
System.Linq.ParallelQuery<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source ParallelQuery<TSource>

A sequence to create a List<T> from.

Returns

List<TSource>

A List<T> that contains elements from the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource> (ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.ToLookup Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource>.

Overloads

Expand table

ToLookup<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function, a comparer and an element selector function.
ToLookup<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToLookup<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function.

ToLookup<TSource,TKey,TElement> (ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function, a comparer and an element selector function.

C#

```
public static System.Linq.ILookup<TKey,TElement>
ToLookup<TSource,TKey,TElement> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector, Func<TSource,TElement>
elementSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` ParallelQuery<TSource>

The sequence to create a ILookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

ILookup<TKey,TElement>

A Lookup<(Of <(TKey, TElement>)>) that contains values of type TElement selected from the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` or `elementSelector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)

Source: ParallelEnumerable.cs

Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to specified key selector and element selector functions.

```
C#

public static System.Linq.ILookup<TKey,TElement>
ToLookup<TSource,TKey,TElement> (this System.Linq.ParallelQuery<TSource>
source, Func<TSource,TKey> keySelector, Func<TSource,TElement>
elementSelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

`TElement`

The type of the value returned by `elementSelector`.

Parameters

`source` ParallelQuery<TSource>

The sequence to create a ILookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`elementSelector` Func<TSource,TElement>

A transform function to produce a result element value from each element.

Returns

ILookup<TKey,TElement>

A ILookup<TKey,TElement> that contains values of type `TElement` selected from the input sequence.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` or `elementSelector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)

Source: ParallelEnumerable.cs

Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.

C#

```
public static System.Linq.ILookup<TKey,TSource> ToLookup<TSource,TKey>
(this System.Linq.ParallelQuery<TSource> source, Func<TSource,TKey>
```

```
keySelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

The sequence to create a ILookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

ILookup<TKey,TSource>

A ILookup<TKey,TElement> that contains keys and values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` or is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ToLookup<TSource,TKey> (ParallelQuery<TSource>, Func<TSource,TKey>)

Source: ParallelEnumerable.cs

Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function.

C#

```
public static System.Linq.ILookup<TKey,TSource> ToLookup<TSource,TKey>
(this System.Linq.ParallelQuery<TSource> source, Func<TSource,TKey>
keySelector);
```

Type Parameters

`TSource`

The type of elements of `source`.

`TKey`

The type of the key returned by `keySelector`.

Parameters

`source` ParallelQuery<TSource>

The sequence to create a ILookup<TKey,TElement> from.

`keySelector` Func<TSource,TKey>

A function to extract a key from each element.

Returns

ILookup<TKey,TSource>

A ILookup<TKey,TElement> that contains keys and values.

Exceptions

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

ArgumentNullException

`source` or `keySelector` is a null reference (Nothing in Visual Basic).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Union Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Produces the set union of two parallel sequences.

Overloads

Expand table

Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set union of two parallel sequences by using the default equality comparer.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set union of two parallel sequences by using a specified IEqualityComparer{T}.

Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TSource> Union<TSource> (this System.Linq.ParallelQuery<TSource> first, System.Collections.Generic.IEnumerable<TSource> second);
```

Type Parameters

TSource

This type parameter is not used.

Parameters

first ParallelQuery<TSource>

This parameter is not used.

second IEnumerable<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

This overload exists to disallow usage of Union with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Union operator would appear to be binding to the parallel implementation, but would in reality bind to sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)

Source: ParallelEnumerable.cs

Produces the set union of two parallel sequences by using the default equality comparer.

```
C#

public static System.Linq.ParallelQuery<TSource> Union<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence whose distinct elements form the first set for the union.

`second` ParallelQuery<TSource>

A sequence whose distinct elements form the second set for the union.

Returns

ParallelQuery<TSource>

A sequence that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
```

```
public static System.Linq.ParallelQuery<TSource> Union<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Collections.Generic.IEnumerable<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

This type parameter is not used.

Parameters

`first` ParallelQuery<TSource>

This parameter is not used.

`second` IEnumerable<TSource>

This parameter is not used.

`comparer` IEqualityComparer<TSource>

This parameter is not used.

Returns

ParallelQuery<TSource>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException

The exception that occurs when this method is called.

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Remarks

This overload exists to disallow usage of Union with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Union operator would appear to be binding to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)

Source: ParallelEnumerable.cs

Produces the set union of two parallel sequences by using a specified IEqualityComparer{T}.

C#

```
public static System.Linq.ParallelQuery<TSource> Union<TSource> (this
System.Linq.ParallelQuery<TSource> first,
System.Linq.ParallelQuery<TSource> second,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`first` ParallelQuery<TSource>

A sequence whose distinct elements form the first set for the union.

`second` ParallelQuery<TSource>

A sequence whose distinct elements form the second set for the union.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

ParallelQuery<TSource>

A sequence that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`first` or `second` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

ParallelEnumerable.Where Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Filters in parallel a sequence of values based on a predicate.

Overloads

Expand table

Where<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Filters in parallel a sequence of values based on a predicate.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

Where<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)

Source: ParallelEnumerable.cs

Filters in parallel a sequence of values based on a predicate.

C#

```
public static System.Linq.ParallelQuery<TSource> Where<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,bool> predicate);
```

Type Parameters

`TSource`

The type of the elements of source.

Parameters

`source` ParallelQuery<TSource>

A sequence to filter.

`predicate` Func<TSource,Boolean>

A function to test each element for a condition.

Returns

ParallelQuery<TSource>

A sequence that contains elements from the input sequence that satisfy the condition.

Exceptions

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Where<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)

Source: ParallelEnumerable.cs

Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

C#

```
public static System.Linq.ParallelQuery<TSource> Where<TSource> (this
System.Linq.ParallelQuery<TSource> source, Func<TSource,int,bool>
predicate);
```

Type Parameters

`TSource`

The type of the elements of source.

Parameters

`source` ParallelQuery<TSource>

A sequence to filter.

`predicate` Func<TSource,Int32,Boolean>

A function to test each element for a condition.

Returns

ParallelQuery<TSource>

A sequence that contains elements from the input sequence that satisfy the condition.

Exceptions

ArgumentNullException

`source` or `predicate` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

OverflowException

More than Int32.MaxValue elements are enumerated by the query.

AggregateException

One or more exceptions occurred during the evaluation of the query.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.WithCancellation<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Sets the CancellationToken to associate with the query.

C#

```
public static System.Linq.ParallelQuery<TSource> WithCancellation<TSource>
(this System.Linq.ParallelQuery<TSource> source,
System.Threading.CancellationToken cancellationToken);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A ParallelQuery on which to set the option.

`cancellationToken` CancellationToken

A cancellation token.

Returns

ParallelQuery<TSource>

A ParallelQuery representing the same query as `source`, but with the registered cancellation token.

Exceptions

ObjectDisposedException

The CancellationTokenSource associated with the `cancellationToken` has been disposed.

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

InvalidOperationException

WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken) is used multiple times in the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)
- How to: Cancel a PLINQ Query
- Cancellation

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.WithDegreeOfParallelism<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Sets the degree of parallelism to use in a query. Degree of parallelism is the maximum number of concurrently executing tasks that will be used to process the query.

C#

```
public static System.Linq.ParallelQuery<TSource>
WithDegreeOfParallelism<TSource> (this System.Linq.ParallelQuery<TSource>
source, int degreeOfParallelism);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A ParallelQuery on which to set the limit on the degrees of parallelism.

`degreeOfParallelism` Int32

The degree of parallelism for the query.

Returns

ParallelQuery<TSource>

ParallelQuery representing the same query as source, with the limit on the degrees of parallelism set.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

ArgumentOutOfRangeException

`degreeOfParallelism` is less than 1 or greater than 512.

InvalidOperationException

WithDegreeOfParallelism is used multiple times in the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.WithExecutionMode<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Sets the execution mode of the query.

C#

```
public static System.Linq.ParallelQuery<TSource> WithExecutionMode<TSource>
(this System.Linq.ParallelQuery<TSource> source,
System.Linq.ParallelExecutionMode executionMode);
```

Type Parameters

TSource

The type of elements of `source`.

Parameters

source ParallelQuery<TSource>

A ParallelQuery on which to set the option.

executionMode ParallelExecutionMode

The mode in which to execute the query.

Returns

ParallelQuery<TSource>

ParallelQuery representing the same query as source, but with the registered execution mode.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

ArgumentException

`executionMode` is not a valid ParallelExecutionMode value.

InvalidOperationException

WithExecutionMode is used multiple times in the query.

Remarks

The execution mode specifies whether PLINQ will attempt to fall back to sequential execution if it detects certain query shapes. For more information and examples, see How to: Specify the Execution Mode in PLINQ.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.WithMergeOptions<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelEnumerable.cs

Sets the merge options for this query, which specify how the query will buffer output.

C#

```
public static System.Linq.ParallelQuery<TSource> WithMergeOptions<TSource>
(this System.Linq.ParallelQuery<TSource> source,
System.Linq.ParallelMergeOptions mergeOptions);
```

Type Parameters

`TSource`

The type of elements of `source`.

Parameters

`source` ParallelQuery<TSource>

A ParallelQuery on which to set the option.

`mergeOptions` ParallelMergeOptions

The merge options to set for this query.

Returns

ParallelQuery<TSource>

ParallelQuery representing the same query as source, but with the registered merge options.

Exceptions

ArgumentNullException

`source` is a null reference (Nothing in Visual Basic).

ArgumentException

`mergeOptions` is not a valid ParallelMergeOptions value.

InvalidOperationException

`WithMergeOptions` is used multiple times in the query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)
- Merge Options in PLINQ
- How to: Specify Merge Options in PLINQ

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelEnumerable.Zip Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Merges in parallel two sequences by using the specified predicate function.

Overloads

Expand table

Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	**Obsolete.** This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, ParallelQuery<TSecond>, Func<TFirst,TSecond,TResult>)	Merges in parallel two sequences by using the specified predicate function.

Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)

Source: ParallelEnumerable.cs

Caution

The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the

AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.

This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.

C#

```
[System.Obsolete("The second data source of a binary operator must be of type System.Linq.ParallelQuery<T> rather than System.Collections.Generic.IEnumerable<T>. To fix this problem, use the AsParallel() extension method to convert the right data source to System.Linq.ParallelQuery<T>.")]
public static System.Linq.ParallelQuery<TResult> Zip<TFirst,TSecond,TResult> (this System.Linq.ParallelQuery<TFirst> first, System.Collections.Generic.IEnumerable<TSecond> second, Func<TFirst,TSecond,TResult> resultSelector);
```

Type Parameters

`TFirst`

This type parameter is not used.

`TSecond`

This type parameter is not used.

`TResult`

This type parameter is not used.

Parameters

`first` ParallelQuery<TFirst>

This parameter is not used.

`second` IEnumerable<TSecond>

This parameter is not used.

`resultSelector` Func<TFirst,TSecond,TResult>

This parameter is not used.

Returns

ParallelQuery<TResult>

This overload always throws a NotSupportedException.

Attributes ObsoleteAttribute

Exceptions

NotSupportedException
The exception that occurs when this method is called.

Remarks

This overload exists to disallow usage of Zip with a left data source of type ParallelQuery<TSource> and a right data source of type IEnumerable<T>. Otherwise, the Zip operator would appear to be bind to the parallel implementation, but would in reality bind to the sequential implementation.

See also

- Parallel LINQ (PLINQ)

Applies to

▼ .NET 9 and other versions

Product	Versions *(Obsolete)*
.NET	*(Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9)*
.NET Framework	*(4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1)*
.NET Standard	*(2.0, 2.1)*
UWP	*(10.0)*

Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>,

ParallelQuery<TSecond>, Func<TFirst,TSecond,TResult>)

Source: ParallelEnumerable.cs

Merges in parallel two sequences by using the specified predicate function.

C#

```
public static System.Linq.ParallelQuery<TResult>
Zip<TFirst,TSecond,TResult> (this System.Linq.ParallelQuery<TFirst>
first, System.Linq.ParallelQuery<TSecond> second,
Func<TFirst,TSecond,TResult> resultSelector);
```

Type Parameters

`TFirst`

The type of the elements of the first sequence.

`TSecond`

The type of the elements of the second sequence.

`TResult`

The type of the return elements.

Parameters

`first` ParallelQuery<TFirst>

The first sequence to zip.

`second` ParallelQuery<TSecond>

The second sequence to zip.

`resultSelector` Func<TFirst,TSecond,TResult>

A function to create a result element from two matching elements.

Returns

ParallelQuery<TResult>

A sequence that has elements of type `TResult` that are obtained by performing `resultSelector` pairwise on two sequences. If the sequence lengths are unequal, this

truncates to the length of the shorter sequence.

Exceptions

ArgumentNullException

`first` or `second` or `resultSelector` is a null reference (Nothing in Visual Basic).

OperationCanceledException

The query was canceled with the token passed in through WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken).

AggregateException

One or more exceptions occurred during the evaluation of the query.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelExecutionMode Enum

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

The query execution mode is a hint that specifies how the system should handle performance trade-offs when parallelizing queries.

C#

```
public enum ParallelExecutionMode
```

Inheritance Object → ValueType → Enum → ParallelExecutionMode

Fields

Expand table

Default	0	This is the default setting. PLINQ will examine the query's structure and will only parallelize the query if will likely result in speedup. If the query structure indicates that speedup is not likely to be obtained, then PLINQ will execute the query as an ordinary LINQ to Objects query.
ForceParallelism	1	Parallelize the entire query, even if that means using high-overhead algorithms. Use this flag in cases where you know that parallel execution of the query will result in speedup, but PLINQ in the Default mode would execute it as sequential.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

See also

- Parallel LINQ (PLINQ)
- How to: Specify the Execution Mode in PLINQ
- Understanding Speedup in PLINQ

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelMergeOptions Enum

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Specifies the preferred type of *output merge* to use in a query. In other words, it indicates how PLINQ should merge the results from the various partitions back into a single result sequence. This is a hint only, and may not be respected by the system when parallelizing all queries.

C#

```
public enum ParallelMergeOptions
```

Inheritance Object → ValueType → Enum → ParallelMergeOptions

Fields

Expand table

AutoBuffered	2	Use a merge with output buffers of a size chosen by the system. Results will accumulate into an output buffer before they are available to the consumer of the query.
Default	0	Use the default merge type, which is AutoBuffered.
FullyBuffered	3	Use a merge with full output buffers. The system will accumulate all of the results before making any of them available to the consumer of the query.
NotBuffered	1	Use a merge without output buffers. As soon as result elements have been computed, make that element available to the consumer of the query.

Remarks

Use `NotBuffered` for queries that will be consumed and output as streams, this has the lowest latency between beginning query execution and elements being yielded. For some queries, such as those involving a sort (OrderBy, OrderByDescending), buffering is

essential and a hint of NotBuffered or AutoBuffered will be ignored. However, queries that are created by using the AsOrdered operator can be streamed as long as no further sorting is performed within the query itself.

Use `AutoBuffered` for most cases; this is the default. It strikes a balance between latency and overall performance.

Use `FullyBuffered` for queries when the entire output can be processed before the information is needed. This option offers the best performance when all of the output can be accumulated before yielding any information, though it is not suitable for stream processing or showing partial results mid-query.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)
- Merge Options in PLINQ
- How to: Specify Merge Options in PLINQ

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelQuery Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelQuery.cs

Represents a parallel sequence.

C#

```
public class ParallelQuery : System.Collections.IEnumerable
```

Inheritance Object → ParallelQuery

Derived System.Linq.ParallelQuery<TSource>

Implements IEnumerable

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Explicit Interface Implementations

Expand table

IEnumerable.GetEnumerator()	Returns an enumerator that iterates through the sequence.

Extension Methods

Expand table

Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
AsOrdered(ParallelQuery)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on non-generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsParallel(IEnumerable)	Enables parallelization of a query.
Cast<TResult>(ParallelQuery)	Converts the elements of a ParallelQuery to the specified type.
OfType<TResult>(ParallelQuery)	Filters the elements of a ParallelQuery based on a specified type.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- ParallelEnumerable

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelQuery.IEnumerable.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelQuery.cs

Returns an enumerator that iterates through the sequence.

```
C#

System.Collections.IEnumerator IEnumerable.GetEnumerator ();
```

Returns

IEnumerator

An enumerator that iterates through the sequence.

Implements

GetEnumerator()

Remarks

This member is an explicit interface member implementation. It can be used only when the ParallelQuery instance is cast to an IEnumerable interface.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

ParallelQuery<TSource> Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelQuery.cs

Represents a parallel sequence.

C#

```
public class ParallelQuery<TSource> : System.Linq.ParallelQuery,
System.Collections.Generic.IEnumerable<TSource>
```

Type Parameters

TSource

The type of element in the source sequence.

Inheritance Object → ParallelQuery → ParallelQuery<TSource>

Derived System.Linq.OrderedParallelQuery<TSource>

Implements IEnumerable<TSource> , IEnumerable

Methods

Expand table

Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetEnumerator()	Returns an enumerator that iterates through the sequence.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)

MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ToString()	Returns a string that represents the current object. (Inherited from Object)

Explicit Interface Implementations

Expand table

IEnumerable.GetEnumerator()	Returns an enumerator that iterates through the sequence. (Inherited from ParallelQuery)

Extension Methods

Expand table

ToFrozenDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector function.
ToFrozenDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a FrozenDictionary<TKey,TValue> from an IEnumerable<T> according to specified key selector and element selector functions.
ToFrozenSet<T>(IEnumerable<T>, IEqualityComparer<T>)	Creates a FrozenSet<T> with the specified values.
ToImmutableArray<TSource>(IEnumerable<TSource>)	Creates an immutable array from the specified collection.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Constructs an immutable dictionary from an existing collection of elements, applying a transformation function to the source keys.
ToImmutableDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Constructs an immutable dictionary based on some

	transformation of a sequence.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key comparer.
ToImmutableDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IEqualityComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable dictionary of its contents by using the specified key and value comparers.
ToImmutableHashSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable hash set of its contents.
ToImmutableHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Enumerates a sequence, produces an immutable hash set of its contents, and uses the specified equality comparer for the set type.
ToImmutableList<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable list of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the specified key comparer.
ToImmutableSortedDictionary<TSource,TKey,TValue>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TValue>, IComparer<TKey>, IEqualityComparer<TValue>)	Enumerates and transforms a sequence, and produces an immutable sorted dictionary of its contents by using the

	specified key and value comparers.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>)	Enumerates a sequence and produces an immutable sorted set of its contents.
ToImmutableSortedSet<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Enumerates a sequence, produces an immutable sorted set of its contents, and uses the specified comparer.
CopyToDataTable<T>(IEnumerable<T>)	Returns a DataTable that contains copies of the DataRow objects, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
CopyToDataTable<T>(IEnumerable<T>, DataTable, LoadOption, FillErrorEventHandler)	Copies DataRow objects to the specified DataTable, given an input IEnumerable<T> object where the generic parameter `T` is DataRow.
Aggregate<TSource>(IEnumerable<TSource>, Func<TSource,TSource,TSource>)	Applies an accumulator function over a sequence.
Aggregate<TSource,TAccumulate>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(IEnumerable<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.

All<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether all elements of a sequence satisfy a condition.
Any<TSource>(IEnumerable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Determines whether any element of a sequence satisfies a condition.
Append<TSource>(IEnumerable<TSource>, TSource)	Appends a value to the end of the sequence.
AsEnumerable<TSource>(IEnumerable<TSource>)	Returns the input typed as IEnumerable<T>.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Cast<TResult>(IEnumerable)	Casts the elements of an IEnumerable to the specified type.
Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of size at most `size`.
Concat<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IEnumerable<TSource>, TSource)	Determines whether a sequence contains a

	specified element by using the default equality comparer.
Contains<TSource>(IEnumerable<TSource>, TSource, IEquality Comparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IEnumerable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified sequence satisfy a condition.
DefaultIfEmpty<TSource>(IEnumerable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IEnumerable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IEnumerable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IEnumerable<TSource>, IEquality Comparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function and using a

	specified comparer to compare keys.
ElementAt<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IEnumerable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEquality Comparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence.
First<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element in a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IEnumerable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.

FirstOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the first element of a sequence, or a specified default value if the sequence contains no elements.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the first element of the sequence that satisfies a condition, or a specified default value if no such element is found.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.
GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

GroupBy<TSource,TKey,TResult>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult> (IEnumerable<TSource>, Func<TSource, TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEquality Comparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.

Intersect<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TKey>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IEnumerable<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence.
Last<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IEnumerable<TSource>)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the last element of a sequence, or a specified default value if the sequence contains no elements.
LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the last element of a sequence that satisfies a condition, or a specified default value if no such element is found.
LongCount<TSource>(IEnumerable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a sequence satisfy a condition.
Max<TSource>(IEnumerable<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic sequence.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a sequence and returns the maximum Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the maximum Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the maximum Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the maximum Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the

	maximum nullable Double value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the maximum nullable Single value.
Max<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a sequence and returns the maximum Single value.
Max<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function.
MaxBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the maximum value in a generic sequence according to a specified key selector function and key comparer.
Min<TSource>(IEnumerable<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic sequence.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Invokes a transform function on each element of a

	sequence and returns the minimum Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Invokes a transform function on each element of a sequence and returns the minimum Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Invokes a transform function on each element of a sequence and returns the minimum Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Invokes a transform function on each element of a sequence and returns the minimum Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.
Min<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Invokes a transform function on each element of a

	sequence and returns the minimum Single value.
Min<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function.
MinBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Returns the minimum value in a generic sequence according to a specified key selector function and key comparer.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
Order<T>(IEnumerable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IEnumerable<T>)	Sorts the elements of a sequence in descending order.

OrderDescending<T>(IEnumerable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IEnumerable<TSource>, TSource)	Adds a value to the beginning of the sequence.
Reverse<TSource>(IEnumerable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Projects each element of a sequence into a new form.
Select<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,TResult>)	Projects each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult> (IEnumerable<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate

	projected form of that element.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer<T>.
Single<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IEnumerable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, TSource)	Returns the only element of a sequence, or a specified default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	Returns the only element of a sequence that satisfies a specified condition, or a specified default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the elements from `source` with the last `count` elements of the source collection omitted.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Decimal>)	Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Double>)	Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int32>)	Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Int64>)	Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Double>>)	Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int32>>)	Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Int64>>)	Computes the sum of the sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(IEnumerable<TSource>, Func<TSource,Nullable<Single>>)	Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.

Sum<TSource>(IEnumerable<TSource>, Func<TSource,Single>)	Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(IEnumerable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IEnumerable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IEnumerable<TSource>, Int32)	Returns a new enumerable collection that contains the last `count` elements from `source`.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(IEnumerable<TSource>)	Creates an array from a IEnumerable<T>.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function.
ToDictionary<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to specified key

	selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from an IEnumerable<T> according to a specified key selector function, a comparer, and an element selector function.
ToHashSet<TSource>(IEnumerable<TSource>)	Creates a HashSet<T> from an IEnumerable<T>.
ToHashSet<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)	Creates a HashSet<T> from an IEnumerable<T> using the `comparer` to compare keys.
ToList<TSource>(IEnumerable<TSource>)	Creates a List<T> from an IEnumerable<T>.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function.
ToLookup<TSource,TKey>(IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(IEnumerable<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according to a specified key selector function, a comparer and an element selector function.

TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IEnumerable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IEnumerable<TSource>, IEnumerable<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>)	Filters a sequence of values based on a predicate.
Where<TSource>(IEnumerable<TSource>, Func<TSource,Int32,Boolean>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.
Zip<TFirst,TSecond,TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond,TResult>(IEnumerable<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.
Aggregate<TSource>(ParallelQuery<TSource>, Func<TSource,TSource,TSource>)	Applies in parallel an accumulator function over a sequence.

Aggregate<TSource,TAccumulate>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, TAccumulate, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate,TResult>(ParallelQuery<TSource>, Func<TAccumulate>, Func<TAccumulate,TSource,TAccumulate>, Func<TAccumulate,TAccumulate,TAccumulate>, Func<TAccumulate,TResult>)	Applies in parallel an accumulator function over a sequence. This overload is not available in the sequential implementation.
All<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether all elements of a sequence satisfy a condition.
Any<TSource>(ParallelQuery<TSource>)	Determines whether a parallel sequence contains any elements.
Any<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Determines in parallel whether any element of a sequence satisfies a condition.
AsEnumerable<TSource>(ParallelQuery<TSource>)	Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.
AsOrdered(ParallelQuery)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on non-generic sequences returned

	by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsOrdered<TSource>(ParallelQuery<TSource>)	Enables treatment of a data source as if it were ordered, overriding the default of unordered. AsOrdered may only be invoked on generic sequences returned by AsParallel, ParallelEnumerable.Range, and ParallelEnumerable.Repeat.
AsParallel(IEnumerable)	Enables parallelization of a query.
AsParallel<TSource>(IEnumerable<TSource>)	Enables parallelization of a query.
AsSequential<TSource>(ParallelQuery<TSource>)	Converts a ParallelQuery<TSource> into an IEnumerable<T> to force sequential evaluation of the query.
AsUnordered<TSource>(ParallelQuery<TSource>)	Allows an intermediate query to be treated as if no ordering is implied among the elements.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function

	on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Average<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the average of a sequence of values that are obtained by invoking a transform function

	on each element of the input sequence.
Cast<TResult>(ParallelQuery)	Converts the elements of a ParallelQuery to the specified type.
Concat<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Concat overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Concat<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Concatenates two parallel sequences.
Contains<TSource>(ParallelQuery<TSource>, TSource)	Determines in parallel whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(ParallelQuery<TSource>, TSource, IEqualityComparer<TSource>)	Determines in parallel whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(ParallelQuery<TSource>)	Returns the number of elements in a parallel sequence.
Count<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns a number that represents how many elements in the specified parallel sequence satisfy a condition.
DefaultIfEmpty<TSource>(ParallelQuery<TSource>)	Returns the elements of the specified parallel sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(ParallelQuery<TSource>, TSource)	Returns the elements of the specified parallel sequence or the specified value in a

	singleton collection if the sequence is empty.
Distinct<TSource>(ParallelQuery<TSource>)	Returns distinct elements from a parallel sequence by using the default equality comparer to compare values.
Distinct<TSource>(ParallelQuery<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a parallel sequence by using a specified IEqualityComparer<T> to compare values.
ElementAt<TSource>(ParallelQuery<TSource>, Int32)	Returns the element at a specified index in a parallel sequence.
ElementAtOrDefault<TSource>(ParallelQuery<TSource>, Int32)	Returns the element at a specified index in a parallel sequence or a default value if the index is out of range.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Except overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set difference of two parallel sequences by using the default equality comparer to compare values.
Except<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two parallel sequences by using the specified IEqualityComparer<T> to compare values.

First<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence.
First<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element in a parallel sequence that satisfies a specified condition.
FirstOrDefault<TSource>(ParallelQuery<TSource>)	Returns the first element of a parallel sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the first element of the parallel sequence that satisfies a condition or a default value if no such element is found.
ForAll<TSource>(ParallelQuery<TSource>, Action<TSource>)	Invokes in parallel the specified action for each element in the `source`.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and compares the keys by using a specified IComparer<T>.
GroupBy<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Groups in parallel the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Groups in parallel the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.

GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TKey,IEnumerable<TSource>,TResult>, IEqualityComparer<TKey>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.
GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>,TResult>)	Groups in parallel the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, Func<TKey,IEnumerable<TElement>, TResult>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	**Obsolete.** This GroupJoin overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>,	**Obsolete.** This GroupJoin overload should never be called. This

Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	method is marked as obsolete and always throws NotSupportedException when called.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>)	Correlates in parallel the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,IEnumerable<TInner>, TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Intersect overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set intersection of two parallel sequences by using the default equality comparer to compare values.
Intersect<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two parallel sequences by using the specified IEqualityComparer{T} to compare values.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>,	**Obsolete.**

Func<TOuter,TInner,TResult>)	This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, IEnumerable<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	**Obsolete.** This Join overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>)	Correlates in parallel the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(ParallelQuery<TOuter>, ParallelQuery<TInner>, Func<TOuter,TKey>, Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	Correlates in parallel the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence.
Last<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a specified condition.
LastOrDefault<TSource>(ParallelQuery<TSource>)	Returns the last element of a parallel sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the last element of a parallel sequence that satisfies a condition, or a default value if no such element is found.
LongCount<TSource>(ParallelQuery<TSource>)	Returns an Int64 that represents the total number of elements in a parallel sequence.

LongCount<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns an Int64 that represents how many elements in a parallel sequence satisfy a condition.
Max<TSource>(ParallelQuery<TSource>)	Returns the maximum value in a parallel sequence of values.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each

	element of a sequence and returns the maximum value.
Max<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Max<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the maximum value.
Min<TSource>(ParallelQuery<TSource>)	Returns the minimum value in a parallel sequence of values.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.

Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
Min<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Invokes in parallel a transform function on each element of a sequence and returns the minimum value.
OfType<TResult>(ParallelQuery)	Filters the elements of a ParallelQuery based on a specified type.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in ascending order according to a key.
OrderBy<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts in parallel the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Sorts in parallel the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
Reverse<TSource>(ParallelQuery<TSource>)	Inverts the order of the elements in a parallel sequence.
Select<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,TResult>)	Projects in parallel each element of a sequence into a new form.

Select<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,TResult>)	Projects in parallel each element of a sequence into a new form by incorporating the element's index.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence.
SelectMany<TSource,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TResult>>)	Projects in parallel each element of a sequence to an IEnumerable<T>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.
SelectMany<TSource,TCollection,TResult>(ParallelQuery<TSource>, Func<TSource,Int32,IEnumerable<TCollection>>, Func<TSource,TCollection,TResult>)	Projects each element of a sequence to an IEnumerable<T>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.
SequenceEqual<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.

SequenceEqual<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This SequenceEqual overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
SequenceEqual<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Determines whether two parallel sequences are equal by comparing the elements by using the default equality comparer for their type.
SequenceEqual<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Determines whether two parallel sequences are equal by comparing their elements by using a specified IEqualityComparer{T}.
Single<TSource>(ParallelQuery<TSource>)	Returns the only element of a parallel sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(ParallelQuery<TSource>)	Returns the only element of a parallel sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns the only element of a parallel sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

Skip<TSource>(ParallelQuery<TSource>, Int32)	Bypasses a specified number of elements in a parallel sequence and then returns the remaining elements.
SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Bypasses elements in a parallel sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Decimal>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Double>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int32>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Int64>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Decimal>>)	Computes in parallel the sum of the sequence of values

	that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Double>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int32>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Int64>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Nullable<Single>>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Sum<TSource>(ParallelQuery<TSource>, Func<TSource,Single>)	Computes in parallel the sum of the sequence of values that are obtained by invoking a transform function on each element of the input sequence.
Take<TSource>(ParallelQuery<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a parallel sequence.
TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Returns elements from a parallel sequence as long as a specified condition is true.
TakeWhile<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Returns elements from a parallel sequence as long as

	a specified condition is true. The element's index is used in the logic of the predicate function.
ToArray<TSource>(ParallelQuery<TSource>)	Creates an array from a ParallelQuery<TSource>.
ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function.
ToDictionary<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToDictionary<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEquality Comparer<TKey>)	Creates a Dictionary<TKey,TValue> from a ParallelQuery<TSource> according to a specified key selector function, a comparer, and an element selector function.
ToList<TSource>(ParallelQuery<TSource>)	Creates a List<T> from an ParallelQuery<TSource>.
ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function.

ToLookup<TSource,TKey>(ParallelQuery<TSource>, Func<TSource,TKey>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function and key comparer.
ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to specified key selector and element selector functions.
ToLookup<TSource,TKey,TElement>(ParallelQuery<TSource>, Func<TSource,TKey>, Func<TSource,TElement>, IEqualityComparer<TKey>)	Creates an ILookup<TKey,TElement> from a ParallelQuery<TSource> according to a specified key selector function, a comparer and an element selector function.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	**Obsolete.** This Union overload should never be called. This method is marked as obsolete and always throws NotSupportedException when called.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>)	Produces the set union of two parallel sequences by using the default equality comparer.
Union<TSource>(ParallelQuery<TSource>, ParallelQuery<TSource>, IEqualityComparer<TSource>)	Produces the set union of two parallel sequences by

	using a specified IEqualityComparer{T}.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Boolean>)	Filters in parallel a sequence of values based on a predicate.
Where<TSource>(ParallelQuery<TSource>, Func<TSource,Int32,Boolean>)	Filters in parallel a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
WithCancellation<TSource>(ParallelQuery<TSource>, CancellationToken)	Sets the CancellationToken to associate with the query.
WithDegreeOfParallelism<TSource>(ParallelQuery<TSource>, Int32)	Sets the degree of parallelism to use in a query. Degree of parallelism is the maximum number of concurrently executing tasks that will be used to process the query.
WithExecutionMode<TSource>(ParallelQuery<TSource>, ParallelExecutionMode)	Sets the execution mode of the query.
WithMergeOptions<TSource>(ParallelQuery<TSource>, ParallelMergeOptions)	Sets the merge options for this query, which specify how the query will buffer output.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, IEnumerable<TSecond>, Func<TFirst,TSecond,TResult>)	**Obsolete.** This Zip overload should never be called. This method is marked as obsolete and always throws NotSupportedException when invoked.
Zip<TFirst,TSecond,TResult>(ParallelQuery<TFirst>, ParallelQuery<TSecond>, Func<TFirst,TSecond,TResult>)	Merges in parallel two sequences by using the specified predicate function.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Ancestors<T>(IEnumerable<T>)	Returns a collection of elements that contains the

	ancestors of every node in the source collection.
Ancestors<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the ancestors of every node in the source collection. Only elements that have a matching XName are included in the collection.
DescendantNodes<T>(IEnumerable<T>)	Returns a collection of the descendant nodes of every document and element in the source collection.
Descendants<T>(IEnumerable<T>)	Returns a collection of elements that contains the descendant elements of every element and document in the source collection.
Descendants<T>(IEnumerable<T>, XName)	Returns a filtered collection of elements that contains the descendant elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
Elements<T>(IEnumerable<T>)	Returns a collection of the child elements of every element and document in the source collection.
Elements<T>(IEnumerable<T>, XName)	Returns a filtered collection of the child elements of every element and document in the source collection. Only elements that have a matching XName are included in the collection.
InDocumentOrder<T>(IEnumerable<T>)	Returns a collection of nodes that contains all nodes in the source collection, sorted in document order.
Nodes<T>(IEnumerable<T>)	Returns a collection of the child nodes of every

	document and element in the source collection.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- ParallelEnumerable
- Parallel LINQ (PLINQ)

ParallelQuery<TSource>.GetEnumerator Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Parallel.dll

Source: ParallelQuery.cs

Returns an enumerator that iterates through the sequence.

C#

```
public virtual System.Collections.Generic.IEnumerator<TSource> GetEnumerator
();
```

Returns

IEnumerator<TSource>

An enumerator that iterates through the sequence.

Implements

GetEnumerator()

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Parallel LINQ (PLINQ)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project.
Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Queryable Class

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Provides a set of `static` (`Shared` in Visual Basic) methods for querying data structures that implement IQueryable<T>.

```
C#
public static class Queryable
```

Inheritance Object → Queryable

Remarks

The set of methods declared in the Queryable class provides an implementation of the standard query operators for querying data sources that implement IQueryable<T>. The standard query operators are general purpose methods that follow the LINQ pattern and enable you to express traversal, filter, and projection operations over data in any .NET-based programming language.

The majority of the methods in this class are defined as extension methods that extend the IQueryable<T> type. This means they can be called like an instance method on any object that implements IQueryable<T>. These methods that extend IQueryable<T> do not perform any querying directly. Instead, their functionality is to build an Expression object, which is an expression tree that represents the cumulative query. The methods then pass the new expression tree to either the Execute<TResult>(Expression) method or the CreateQuery<TElement>(Expression) method of the input IQueryable<T>. The method that is called depends on whether the Queryable method returns a singleton value, in which case Execute<TResult>(Expression) is called, or has enumerable results, in which case CreateQuery<TElement>(Expression) is called.

The actual query execution on the target data is performed by a class that implements IQueryable<T>. The expectation of any IQueryable<T> implementation is that the result

of executing an expression tree that was constructed by a Queryable standard query operator method is equivalent to the result of calling the corresponding method in the Enumerable class, if the data source were an IEnumerable<T>.

In addition to the standard query operator methods that operate on IQueryable<T> objects, this class also contains a method, AsQueryable, which types IEnumerable objects as IQueryable objects.

Methods

Expand table

Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>)	Applies an accumulator function over a sequence.
All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether all the elements of a sequence satisfy a condition.
Any<TSource>(IQueryable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether any element of a sequence satisfies a condition.

Append<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` appended at the end.
AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement>(IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.
Average(IQueryable<Decimal>)	Computes the average of a sequence of Decimal values.
Average(IQueryable<Double>)	Computes the average of a sequence of Double values.
Average(IQueryable<Int32>)	Computes the average of a sequence of Int32 values.
Average(IQueryable<Int64>)	Computes the average of a sequence of Int64 values.
Average(IQueryable<Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values.
Average(IQueryable<Nullable<Double>>)	Computes the average of a sequence of nullable Double values.
Average(IQueryable<Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values.
Average(IQueryable<Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values.

Average(IQueryable<Nullable<Single>>)	Computes the average of a sequence of nullable Single values.
Average(IQueryable<Single>)	Computes the average of a sequence of Single values.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the average of a sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the average of a sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the average of a sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the average of a sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the average of a sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the average of a sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the average of a sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the average of a sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the average of a sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the average of a sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Cast<TResult>(IQueryable)	Converts the elements of an IQueryable to the specified type.
Chunk<TSource>(IQueryable<TSource>, Int32)	Split the elements of a sequence into chunks

	of size at most `size`.
Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Concatenates two sequences.
Contains<TSource>(IQueryable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.
Count<TSource>(IQueryable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the number of elements in the specified sequence that satisfies a condition.
DefaultIfEmpty<TSource>(IQueryable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
Distinct<TSource>(IQueryable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a

	specified IEqualityComparer<T> to compare values.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function.
ElementAt<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.

ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
First<TSource>(IQueryable<TSource>)	Returns the first element of a sequence.
First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition.
FirstOrDefault<TSource>(IQueryable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.
FirstOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value

	from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>)	Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEquality Comparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and

	creates a result value from each group and its key. Keys are compared by using a specified comparer.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>)	Correlates the elements of two sequences based on key equality and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified

	IEqualityComparer<T> to compare values.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.
Last<TSource>(IQueryable<TSource>)	Returns the last element in a sequence.
Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a specified condition.
LastOrDefault<TSource>(IQueryable<TSource>)	Returns the last element in a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a condition or a default

	value if no such element is found.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LongCount<TSource>(IQueryable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns an Int64 that represents the number of elements in a sequence that satisfy a condition.
Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the maximum resulting value.
Max<TSource>(IQueryable<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T>.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the maximum value in a generic

	IQueryable<T> according to a specified key selector function.
Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the minimum resulting value.
Min<TSource>(IQueryable<TSource>)	Returns the minimum value of a generic IQueryable<T>.
Min<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T>.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
OfType<TResult>(IQueryable)	Filters the elements of an IQueryable based on a specified type.
Order<T>(IQueryable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in ascending order according to a key.

OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in descending order according to a key.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderDescending<T>(IQueryable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.
Prepend<TSource>(IQueryable<TSource>, TSource)	Returns a new queryable sequence that contains the elements from `source` plus the specified `element` prepended at the beginning.
Reverse<TSource>(IQueryable<TSource>)	Inverts the order of the elements in a sequence.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>)	Projects each element of a sequence into a new form by incorporating the element's index.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Projects each element of a sequence into a new form.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> and invokes a result selector function on

	each element therein. The resulting values from each intermediate sequence are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> that incorporates the index of the source element that produced it. A result selector function is invoked on each element of each intermediate sequence, and the resulting values are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by using the default equality comparer to compare elements.

SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by using a specified IEqualityComparer<T> to compare elements.
Single<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.
SingleOrDefault<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this

	method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
Skip<TSource>(IQueryable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
SkipLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the elements from `source` with the last `count` elements of the source queryable sequence omitted.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.
Sum(IQueryable<Decimal>)	Computes the sum of a sequence of Decimal values.

Sum(IQueryable<Double>)	Computes the sum of a sequence of Double values.
Sum(IQueryable<Int32>)	Computes the sum of a sequence of Int32 values.
Sum(IQueryable<Int64>)	Computes the sum of a sequence of Int64 values.
Sum(IQueryable<Nullable<Decimal>>)	Computes the sum of a sequence of nullable Decimal values.
Sum(IQueryable<Nullable<Double>>)	Computes the sum of a sequence of nullable Double values.
Sum(IQueryable<Nullable<Int32>>)	Computes the sum of a sequence of nullable Int32 values.
Sum(IQueryable<Nullable<Int64>>)	Computes the sum of a sequence of nullable Int64 values.
Sum(IQueryable<Nullable<Single>>)	Computes the sum of a sequence of nullable Single values.
Sum(IQueryable<Single>)	Computes the sum of a sequence of Single values.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the sum of the sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the sum of the sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the sum of the sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the sum of the sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the sum of the sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the sum of the sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the sum of the sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the sum of the sequence of nullable Int64 values that is obtained by invoking a projection function on each

	element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the sum of the sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the sum of the sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Take<TSource>(IQueryable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource>(IQueryable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.
TakeLast<TSource>(IQueryable<TSource>, Int32)	Returns a new queryable sequence that contains the last `count` elements from `source`.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns elements from a sequence as long as a specified condition is true.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the

	elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.
ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Filters a sequence of values based on a predicate.

Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
Zip<TFirst,TSecond,TResult>(IQueryable<TFirst>, IEnumerable<TSecond>, Expression<Func<TFirst,TSecond,TResult>>)	Merges two sequences by using the specified predicate function.
Zip<TFirst,TSecond,TThird>(IQueryable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond>(IQueryable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

See also

- Language-Integrated Query (LINQ)
- Standard Query Operators Overview
- Expression Trees
- LINQ to SQL

Queryable.Aggregate Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.
Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>)	Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.
Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>)	Applies an accumulator function over a sequence.

Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>)

Source: Queryable.cs

Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result

value.

```
C#

public static TResult Aggregate<TSource,TAccumulate,TResult> (this
System.Linq.IQueryable<TSource> source, TAccumulate seed,
System.Linq.Expressions.Expression<Func<TAccumulate,TSource,TAccumulate>>
func, System.Linq.Expressions.Expression<Func<TAccumulate,TResult>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

`TResult`

The type of the resulting value.

Parameters

`source` IQueryable<TSource>

A sequence to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`func` Expression<Func<TAccumulate,TSource,TAccumulate>>

An accumulator function to invoke on each element.

`selector` Expression<Func<TAccumulate,TResult>>

A function to transform the final accumulator value into the result value.

Returns

TResult

The transformed final accumulator value.

Exceptions

ArgumentNullException

`source` or `func` or `selector` is `null`.

Examples

The following code example demonstrates how to use Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>) to apply an accumulator function and a result selector.

C#

```
string[] fruits = { "apple", "mango", "orange", "passionfruit", "grape"
};

// Determine whether any string in the array is longer than "banana".
string longestName =
    fruits.AsQueryable().Aggregate(
    "banana",
    (longest, next) => next.Length > longest.Length ? next : longest,
    // Return the final result as an uppercase string.
    fruit => fruit.ToUpper()
    );

Console.WriteLine(
    "The fruit with the longest name is {0}.",
    longestName);

// This code produces the following output:
//
// The fruit with the longest name is PASSIONFRUIT.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>) method generates a MethodCallExpression that represents calling Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>,

Expression<Func<TAccumulate, TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Aggregate<TSource,TAccumulate,TResult>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>, Expression<Func<TAccumulate, TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that the specified function, `func`, is applied to each value in the source sequence and the accumulated value is returned. The `seed` parameter is used as the seed value for the accumulated value, which corresponds to the first parameter in `func`. The final accumulated value is passed to `selector` to obtain the result value.

To simplify common aggregation operations, the set of standard query operators also includes two counting methods, Count and LongCount, and four numeric aggregation methods, namely Max, Min, Sum, and Average.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>)

Source: Queryable.cs

Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.

C#

```
public static TAccumulate Aggregate<TSource,TAccumulate> (this
System.Linq.IQueryable<TSource> source, TAccumulate seed,
System.Linq.Expressions.Expression<Func<TAccumulate,TSource,TAccumulate>>
func);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TAccumulate`

The type of the accumulator value.

Parameters

`source` IQueryable<TSource>

A sequence to aggregate over.

`seed` TAccumulate

The initial accumulator value.

`func` Expression<Func<TAccumulate,TSource,TAccumulate>>

An accumulator function to invoke on each element.

Returns

TAccumulate

The final accumulator value.

Exceptions

ArgumentNullException

`source` or `func` is `null`.

Examples

The following code example demonstrates how to use Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>) to apply an accumulator function when a seed value is provided to the function.

```C#
int[] ints = { 4, 8, 8, 3, 9, 0, 7, 8, 2 };

// Count the even numbers in the array, using a seed value of 0.
int numEven =
    ints.AsQueryable().Aggregate(
    0,
    (total, next) => next % 2 == 0 ? total + 1 : total
    );

Console.WriteLine("The number of even integers is: {0}", numEven);

// This code produces the following output:
//
// The number of even integers is: 6
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>) method generates a MethodCallExpression that represents calling Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Aggregate<TSource,TAccumulate>(IQueryable<TSource>, TAccumulate, Expression<Func<TAccumulate,TSource,TAccumulate>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that the specified function, `func`, is applied to each value in the source sequence and the accumulated value is returned. The `seed` parameter is used as the seed value for the accumulated value, which corresponds to the first parameter in `func`.

To simplify common aggregation operations, the set of standard query operators also includes two counting methods, Count and LongCount, and four numeric aggregation methods, namely Max, Min, Sum, and Average.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>)

Source: Queryable.cs

Applies an accumulator function over a sequence.

C#

```
public static TSource Aggregate<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TSource,TSource>> func);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence to aggregate over.

`func` Expression<Func<TSource,TSource,TSource>>

An accumulator function to apply to each element.

Returns

TSource

The final accumulator value.

Exceptions

ArgumentNullException

`source` or `func` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>) to build a sentence from an array of strings.

C#

```
string sentence = "the quick brown fox jumps over the lazy dog";

// Split the string into individual words.
string[] words = sentence.Split(' ');

// Use Aggregate() to prepend each word to the beginning of the
// new sentence to reverse the word order.
string reversed =
    words.AsQueryable().Aggregate(
    (workingSentence, next) => next + " " + workingSentence
    );

Console.WriteLine(reversed);

// This code produces the following output:
//
// dog lazy the over jumps fox brown quick the
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>) method generates a MethodCallExpression that represents calling Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Aggregate<TSource>(IQueryable<TSource>, Expression<Func<TSource,TSource,TSource>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that the specified function, `func`, is applied to each value in the source sequence and the accumulated value is returned. The first value in `source` is used as the seed value for the accumulated value, which corresponds to the first parameter in `func`.

To simplify common aggregation operations, the set of standard query operators also includes two counting methods, Count and LongCount, and four numeric aggregation methods, namely Max, Min, Sum, and Average.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.All<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Determines whether all the elements of a sequence satisfy a condition.

C#

```
public static bool All<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

A sequence whose elements to test for a condition.

predicate Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

Boolean

`true` if every element of the source sequence passes the test in the specified predicate, or if the sequence is empty; otherwise, `false`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to determine whether all the elements in a sequence satisfy a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void AllEx1()
{
    // Create an array of Pets.
    Pet[] pets = { new Pet { Name="Barley", Age=10 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=6 } };

    // Determine whether all pet names in the array start with 'B'.
    bool allStartWithB =
        pets.AsQueryable().All(pet => pet.Name.StartsWith("B"));

    Console.WriteLine(
        "{0} pet names start with 'B'.",
        allStartWithB ? "All" : "Not all");
}

// This code produces the following output:
//
//  Not all pet names start with 'B'.
```

The Boolean value that the All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method returns is typically used in the predicate of a `where` clause (`Where` clause in Visual Basic) or a direct call to the Where method. The following example demonstrates this use of the `All` method.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}
class Person
{
    public string LastName { get; set; }
    public Pet[] Pets { get; set; }
```

```
}

public static void AllEx2()
{
    List<Person> people = new List<Person>
        { new Person { LastName = "Haas",
                       Pets = new Pet[] { new Pet { Name="Barley", Age=10 },
                                          new Pet { Name="Boots", Age=14 },
                                          new Pet { Name="Whiskers", Age=6
}}},
          new Person { LastName = "Fakhouri",
                       Pets = new Pet[] { new Pet { Name = "Snowball", Age =
1}}},
          new Person { LastName = "Antebi",
                       Pets = new Pet[] { new Pet { Name = "Belle", Age = 8}
}},
          new Person { LastName = "Philips",
                       Pets = new Pet[] { new Pet { Name = "Sweetie", Age =
2},
                                          new Pet { Name = "Rover", Age =
13}} }
        };

    // Determine which people have pets that are all older than 5.
    IEnumerable<string> names = from person in people
                                where person.Pets.AsQueryable().All(pet =>
pet.Age > 5)
                                select person.LastName;

    foreach (string name in names)
        Console.WriteLine(name);

    /* This code produces the following output:
     *
     * Haas
     * Antebi
     */
}
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>

(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling All<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the `source` parameter's type. The expected behavior is that it determines if all the elements in `source` satisfy the condition in `predicate`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Any Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Determines whether any element of an IQueryable<T> sequence exists or satisfies a condition.

Overloads

Expand table

Any<TSource>(IQueryable<TSource>)	Determines whether a sequence contains any elements.
Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Determines whether any element of a sequence satisfies a condition.

Any<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Determines whether a sequence contains any elements.

C#

```
public static bool Any<TSource> (this System.Linq.IQueryable<TSource>
source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

A sequence to check for being empty.

Returns

Boolean

`true` if the source sequence contains any elements; otherwise, `false`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Any<TSource> (IQueryable<TSource>) to determine whether a sequence contains any elements.

C#

```
List<int> numbers = new List<int> { 1, 2 };

// Determine if the list contains any elements.
bool hasElements = numbers.AsQueryable().Any();

Console.WriteLine("The list {0} empty.",
    hasElements ? "is not" : "is");

// This code produces the following output:
//
// The list is not empty.
```

The Boolean value that the Any<TSource>(IQueryable<TSource>) method returns is typically used in the predicate of a `where` clause (`Where` clause in Visual Basic) or a direct call to the Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method. The following example demonstrates this use of the `Any` method.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}
class Person
```

```
{
    public string LastName { get; set; }
    public Pet[] Pets { get; set; }
}

public static void AnyEx2()
{
    List<Person> people = new List<Person>
        { new Person { LastName = "Haas",
                       Pets = new Pet[] { new Pet { Name="Barley", Age=10
},
                                          new Pet { Name="Boots", Age=14
},
                                          new Pet { Name="Whiskers",
Age=6 }}},
          new Person { LastName = "Fakhouri",
                       Pets = new Pet[] { new Pet { Name = "Snowball",
Age = 1}}},
          new Person { LastName = "Antebi",
                       Pets = new Pet[] { }},
          new Person { LastName = "Philips",
                       Pets = new Pet[] { new Pet { Name = "Sweetie", Age
= 2},
                                          new Pet { Name = "Rover", Age =
13}} }
        };

    // Determine which people have a non-empty Pet array.
    IEnumerable<string> names = from person in people
                                where person.Pets.AsQueryable().Any()
                                select person.LastName;

    foreach (string name in names)
        Console.WriteLine(name);

    /* This code produces the following output:

       Haas
       Fakhouri
       Philips
    */
}
```

Remarks

The Any<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Any<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Any<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it determines if `source` contains any elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Determines whether any element of a sequence satisfies a condition.

C#

```
public static bool Any<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,bool>>
predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence whose elements to test for a condition.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

Boolean

`true` if any elements in the source sequence pass the test in the specified predicate; otherwise, `false`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use Any<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to determine whether any element in a sequence satisfies a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
    public bool Vaccinated { get; set; }
}

public static void AnyEx3()
{
    // Create an array of Pet objects.
    Pet[] pets =
        { new Pet { Name="Barley", Age=8, Vaccinated=true },
          new Pet { Name="Boots", Age=4, Vaccinated=false },
          new Pet { Name="Whiskers", Age=1, Vaccinated=false } };

    // Determine whether any pets over age 1 are also unvaccinated.
    bool unvaccinated =
        pets.AsQueryable().Any(p => p.Age > 1 && p.Vaccinated == false);

    Console.WriteLine(
        "There {0} unvaccinated animals over age one.",
        unvaccinated ? "are" : "are not any");
}

// This code produces the following output:
```

```
//
//  There are unvaccinated animals over age one.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Any<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it determines if any of the elements of `source` satisfy the condition specified by `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Append<TSource> (IQueryable<TSource>, TSource) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Returns a new queryable sequence that contains the elements from `source` plus the specified `element` appended at the end.

```
C#

public static System.Linq.IQueryable<TSource> Append<TSource> (this
System.Linq.IQueryable<TSource> source, TSource element);
```

Type Parameters

`TSource`

The type of the elements in the queryable sequence.

Parameters

`source` IQueryable<TSource>

A queryable sequence.

`element` TSource

An element of type `TSource` to append to `source`.

Returns

IQueryable<TSource>

A new queryable sequence that contains the elements from `source` plus the specified `element` appended at the end.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Queryable.AsQueryable Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

AsQueryable(IEnumerable)	Converts an IEnumerable to an IQueryable.
AsQueryable<TElement> (IEnumerable<TElement>)	Converts a generic IEnumerable<T> to a generic IQueryable<T>.

AsQueryable(IEnumerable)

Source: Queryable.cs

Converts an IEnumerable to an IQueryable.

C#

```
public static System.Linq.IQueryable AsQueryable (this
System.Collections.IEnumerable source);
```

Parameters

`source` IEnumerable

A sequence to convert.

Returns

IQueryable

An IQueryable that represents the input sequence.

Exceptions

ArgumentException

`source` does not implement IEnumerable<T> for some `T`.

ArgumentNullException

`source` is `null`.

Remarks

If the type of `source` implements IQueryable<T>, AsQueryable(IEnumerable) returns it directly. Otherwise, it returns an IQueryable<T> that executes queries by calling the equivalent query operator methods in Enumerable instead of those in Queryable.

This method assumes that `source` implements IEnumerable<T> for some `T`. At runtime, the result is of type IQueryable<T> for the same `T`. This method is useful in dynamic scenarios when you do not statically know the type of `T`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

AsQueryable<TElement> (IEnumerable<TElement>)

Source: Queryable.cs

Converts a generic IEnumerable<T> to a generic IQueryable<T>.

C#

```
public static System.Linq.IQueryable<TElement> AsQueryable<TElement>
```

```
(this System.Collections.Generic.IEnumerable<TElement> source);
```

Type Parameters

`TElement`

The type of the elements of `source`.

Parameters

`source` IEnumerable<TElement>

A sequence to convert.

Returns

IQueryable<TElement>

An IQueryable<T> that represents the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use AsQueryable<TElement> (IEnumerable<TElement>) to convert an IEnumerable<T> to an IQueryable<T>.

C#

```
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };

// Convert the List to an IQueryable<int>.
IQueryable<int> iqueryable = grades.AsQueryable();

// Get the Expression property of the IQueryable object.
System.Linq.Expressions.Expression expressionTree =
    iqueryable.Expression;

Console.WriteLine("The NodeType of the expression tree is: "
    + expressionTree.NodeType.ToString());
Console.WriteLine("The Type of the expression tree is: "
    + expressionTree.Type.Name);

/*
    This code produces the following output:
```

```
    The NodeType of the expression tree is: Constant
    The Type of the expression tree is: EnumerableQuery`1
*/
```

Remarks

If the type of `source` implements IQueryable<T>, AsQueryable<TElement> (IEnumerable<TElement>) returns it directly. Otherwise, it returns an IQueryable<T> that executes queries by calling the equivalent query operator methods in Enumerable instead of those in Queryable.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Average Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Computes the average of a sequence of numeric values.

Overloads

Expand table

Average(IQueryable<Single>)	Computes the average of a sequence of Single values.
Average(IQueryable<Nullable<Int64>>)	Computes the average of a sequence of nullable Int64 values.
Average(IQueryable<Nullable<Int32>>)	Computes the average of a sequence of nullable Int32 values.
Average(IQueryable<Nullable<Double>>)	Computes the average of a sequence of nullable Double values.
Average(IQueryable<Nullable<Single>>)	Computes the average of a sequence of nullable Single values.
Average(IQueryable<Int64>)	Computes the average of a sequence of Int64 values.
Average(IQueryable<Int32>)	Computes the average of a sequence of Int32 values.
Average(IQueryable<Double>)	Computes the average of a sequence of Double values.
Average(IQueryable<Decimal>)	Computes the average of a sequence of Decimal values.
Average(IQueryable<Nullable<Decimal>>)	Computes the average of a sequence of nullable Decimal values.

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the average of a sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the average of a sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the average of a sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the average of a sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the average of a sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the average of a sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the average of a sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the average of a sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.
Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the average of a sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the average of a sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.

Average(IQueryable<Single>)

Source: Queryable.cs

Computes the average of a sequence of Single values.

```
C#
public static float Average (this System.Linq.IQueryable<float> source);
```

Parameters

`source` IQueryable<Single>

A sequence of Single values to calculate the average of.

Returns

Single

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average(IQueryable<Int32>) to calculate the average of a sequence of values.

ⓘ Note

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

C#

```
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };

double average = grades.AsQueryable().Average();

Console.WriteLine("The average grade is {0}.", average);

// This code produces the following output:
//
// The average grade is 77.6.
```

Remarks

The Average(IQueryable<Single>) method generates a MethodCallExpression that represents calling Average(IQueryable<Single>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Single>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Nullable<Int64>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Int64 values.

```
C#

public static double? Average (this System.Linq.IQueryable<long?>
source);
```

Parameters

source IQueryable<Nullable<Int64>>

A sequence of nullable Int64 values to calculate the average of.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Average(IQueryable<Nullable<Int64>>) to calculate the average of a sequence of values.

```
C#

long?[] longs = { null, 10007L, 37L, 399846234235L };

double? average = longs.AsQueryable().Average();

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
```

```
//
// The average is 133282081426.333.
```

Remarks

The Average(IQueryable<Nullable<Int64>>) method generates a MethodCallExpression that represents calling Average(IQueryable<Nullable<Int64>>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Nullable<Int64>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Nullable<Int32>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Int32 values.

C#

```
public static double? Average (this System.Linq.IQueryable<int?> source);
```

Parameters

`source` IQueryable<Nullable<Int32>>

A sequence of nullable Int32 values to calculate the average of.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Average(IQueryable<Nullable<Int64>>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

C#

```
long?[] longs = { null, 10007L, 37L, 399846234235L };

double? average = longs.AsQueryable().Average();

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 133282081426.333.
```

Remarks

The Average(IQueryable<Nullable<Int32>>) method generates a MethodCallExpression that represents calling Average(IQueryable<Nullable<Int32>>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Nullable<Int32>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Nullable<Double>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Double values.

C#

```
public static double? Average (this System.Linq.IQueryable<double?>
source);
```

Parameters

`source` IQueryable<Nullable<Double>>

A sequence of nullable Double values to calculate the average of.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the source sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Average(IQueryable<Nullable<Int64>>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

C#

```
long?[] longs = { null, 10007L, 37L, 399846234235L };

double? average = longs.AsQueryable().Average();

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 133282081426.333.
```

Remarks

The Average(IQueryable<Nullable<Double>>) method generates a MethodCallExpression that represents calling Average(IQueryable<Nullable<Double>>) itself. It then passes the

MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Nullable<Double>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Nullable<Single>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Single values.

C#

```
public static float? Average (this System.Linq.IQueryable<float?>
source);
```

Parameters

`source` IQueryable<Nullable<Single>>

A sequence of nullable Single values to calculate the average of.

Returns

Nullable<Single>

The average of the sequence of values, or `null` if the source sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Average(IQueryable<Nullable<Int64>>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

C#

```
long?[] longs = { null, 10007L, 37L, 399846234235L };

double? average = longs.AsQueryable().Average();

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
//
// The average is 133282081426.333.
```

Remarks

The Average(IQueryable<Nullable<Single>>) method generates a MethodCallExpression that represents calling Average(IQueryable<Nullable<Single>>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Nullable<Single>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Int64>)

Source: Queryable.cs

Computes the average of a sequence of Int64 values.

C#

```
public static double Average (this System.Linq.IQueryable<long> source);
```

Parameters

`source` IQueryable<Int64>

A sequence of Int64 values to calculate the average of.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average(IQueryable<Int32>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

C#

```
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };

double average = grades.AsQueryable().Average();

Console.WriteLine("The average grade is {0}.", average);

// This code produces the following output:
//
// The average grade is 77.6.
```

Remarks

The Average(IQueryable<Int64>) method generates a MethodCallExpression that represents calling Average(IQueryable<Int64>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Int64>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Int32>)

Source: Queryable.cs

Computes the average of a sequence of Int32 values.

C#

```
public static double Average (this System.Linq.IQueryable<int> source);
```

Parameters

`source` IQueryable<Int32>

A sequence of Int32 values to calculate the average of.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average(IQueryable<Int32>) to calculate the average of a sequence of values.

C#

```
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };

double average = grades.AsQueryable().Average();

Console.WriteLine("The average grade is {0}.", average);

// This code produces the following output:
//
// The average grade is 77.6.
```

Remarks

The Average(IQueryable<Int32>) method generates a MethodCallExpression that represents calling Average(IQueryable<Int32>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Int32>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Double>)

Source: Queryable.cs

Computes the average of a sequence of Double values.

```
C#

public static double Average (this System.Linq.IQueryable<double>
source);
```

Parameters

`source` IQueryable<Double>

A sequence of Double values to calculate the average of.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average(IQueryable<Int32>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

```C#
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };

double average = grades.AsQueryable().Average();

Console.WriteLine("The average grade is {0}.", average);

// This code produces the following output:
//
// The average grade is 77.6.
```

Remarks

The Average(IQueryable<Double>) method generates a MethodCallExpression that represents calling Average(IQueryable<Double>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Double>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Decimal>)

Source: Queryable.cs

Computes the average of a sequence of Decimal values.

```C#
public static decimal Average (this System.Linq.IQueryable<decimal>
source);
```

Parameters

source IQueryable<Decimal>

A sequence of Decimal values to calculate the average of.

Returns

Decimal

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average(IQueryable<Int32>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

```C#
List<int> grades = new List<int> { 78, 92, 100, 37, 81 };
```

```
double average = grades.AsQueryable().Average();

Console.WriteLine("The average grade is {0}.", average);

// This code produces the following output:
//
// The average grade is 77.6.
```

Remarks

The Average(IQueryable<Decimal>) method generates a MethodCallExpression that represents calling Average(IQueryable<Decimal>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Decimal>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average(IQueryable<Nullable<Decimal>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Decimal values.

```
C#
```

```
public static decimal? Average (this System.Linq.IQueryable<decimal?>
source);
```

Parameters

`source` IQueryable<Nullable<Decimal>>

A sequence of nullable Decimal values to calculate the average of.

Returns

Nullable<Decimal>

The average of the sequence of values, or `null` if the source sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Average(IQueryable<Nullable<Int64>>) to calculate the average of a sequence of values.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, substitute the elements of the source sequence with elements of the appropriate numerical type.

C#

```
long?[] longs = { null, 10007L, 37L, 399846234235L };

double? average = longs.AsQueryable().Average();

Console.WriteLine("The average is {0}.", average);

// This code produces the following output:
```

```
//
// The average is 133282081426.333.
```

Remarks

The Average(IQueryable<Nullable<Decimal>>) method generates a MethodCallExpression that represents calling Average(IQueryable<Nullable<Decimal>>) itself. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average(IQueryable<Nullable<Decimal>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)

Source: Queryable.cs

Computes the average of a sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.

```
C#
```

```
public static float Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,float>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Single>>

A projection function to apply to each element.

Returns

Single

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the

overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static float? Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,float?>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Nullable<Single>>>

A projection function to apply to each element.

Returns

Nullable<Single>

The average of the sequence of values, or `null` if the `source` sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>) method generates a MethodCallExpression that represents calling Average<TSource>

(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double? Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,long?>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Nullable<Int64>>>

A projection function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the `source` sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);
```

```
Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double? Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,double?>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Nullable<Double>>>

A projection function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the `source` sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double? Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int?>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Nullable<Int32>>>

A projection function to apply to each element.

Returns

Nullable<Double>

The average of the sequence of values, or `null` if the `source` sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)

Source: Queryable.cs

Computes the average of a sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,long>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Int64>>

A projection function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

```
C#
```

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)

Source: Queryable.cs

Computes the average of a sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Int32>>

A projection function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)

Source: Queryable.cs

Computes the average of a sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,double>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Double>>

A projection function to apply to each element.

Returns

Double

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)

Source: Queryable.cs

Computes the average of a sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static decimal Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,decimal>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values that are used to calculate an average.

`selector` Expression<Func<TSource,Decimal>>

A projection function to apply to each element.

Returns

Decimal

The average of the sequence of values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

```
C#
```

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)

Source: Queryable.cs

Computes the average of a sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.

```
C#

public static decimal? Average<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,decimal?>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to calculate the average of.

`selector` Expression<Func<TSource,Nullable<Decimal>>>

A projection function to apply to each element.

Returns

Nullable<Decimal>

The average of the sequence of values, or `null` if the `source` sequence is empty or contains only `null` values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) to calculate the average String length in a sequence of values of type String.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit",
"grape" };

// Determine the average string length in the array.
double average = fruits.AsQueryable().Average(s => s.Length);

Console.WriteLine("The average string length is {0}.", average);

// This code produces the following output:
//
// The average string length is 6.5.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>) method generates a MethodCallExpression that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Average<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it calculates the average of the values in `source` after invoking `selector` on each value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Cast<TResult>(IQueryable) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Converts the elements of an IQueryable to the specified type.

C#

```
public static System.Linq.IQueryable<TResult> Cast<TResult> (this
System.Linq.IQueryable source);
```

Type Parameters

`TResult`

The type to convert the elements of `source` to.

Parameters

`source` IQueryable

The IQueryable that contains the elements to be converted.

Returns

IQueryable<TResult>

An IQueryable<T> that contains each element of the source sequence converted to the specified type.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidCastException

An element in the sequence cannot be cast to type `TResult`.

Examples

The following code example demonstrates how to use Cast<TResult>(IQueryable) to convert objects in a sequence to type String.

C#

```
// Create a list of objects.
List<object> words =
    new List<object> { "green", "blue", "violet" };

// Cast the objects in the list to type 'string'
// and project the first letter of each string.
IEnumerable<string> query =
    words.AsQueryable()
    .Cast<string>()
    .Select(str => str.Substring(0, 1));

foreach (string s in query)
    Console.WriteLine(s);

/*  This code produces the following output:

    g
    b
    v
*/
```

Remarks

The Cast<TResult>(IQueryable) method generates a MethodCallExpression that represents calling Cast<TResult>(IQueryable) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Cast<TResult>(IQueryable) depends on the implementation of the type of the `source` parameter. The expected behavior is that it converts the values in `source` to type `TResult`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Chunk<TSource> (IQueryable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Split the elements of a sequence into chunks of size at most `size`.

C#

```
public static System.Linq.IQueryable<TSource[]> Chunk<TSource> (this
System.Linq.IQueryable<TSource> source, int size);
```

Type Parameters

`TSource`

The type of the elements of source.

Parameters

`source` IQueryable<TSource>

An IEnumerable<T> whose elements to chunk.

`size` Int32

The maximum size of each chunk.

Returns

IQueryable<TSource[]>

An IEnumerable<T> that contains the elements the input sequence split into chunks of size `size`.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentOutOfRangeException

`size` is below 1.

Remarks

Every chunk except the last will be of size `size`.

The last chunk will contain the remaining elements and may be of a smaller size.

Applies to

Product	Versions
.NET	6, 7, 8, 9

Queryable.Concat<TSource> Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Concatenates two sequences.

C#

```
public static System.Linq.IQueryable<TSource> Concat<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

source1 IQueryable<TSource>

The first sequence to concatenate.

source2 IEnumerable<TSource>

The sequence to concatenate to the first sequence.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the concatenated elements of the two input sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Examples

The following code example demonstrates how to use Concat<TSource> (IQueryable<TSource>, IEnumerable<TSource>) to concatenate two sequences.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

// This method creates and returns an array of Pet objects.
static Pet[] GetCats()
{
    Pet[] cats = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
    return cats;
}

// This method creates and returns an array of Pet objects.
static Pet[] GetDogs()
{
    Pet[] dogs = { new Pet { Name="Bounder", Age=3 },
                   new Pet { Name="Snoopy", Age=14 },
                   new Pet { Name="Fido", Age=9 } };
    return dogs;
}

public static void ConcatEx1()
{
    Pet[] cats = GetCats();
    Pet[] dogs = GetDogs();

    // Concatenate a collection of cat names to a
    // collection of dog names by using Concat().
    IEnumerable<string> query =
        cats.AsQueryable()
        .Select(cat => cat.Name)
        .Concat(dogs.Select(dog => dog.Name));

    foreach (string name in query)
        Console.WriteLine(name);
}

// This code produces the following output:
//
// Barley
// Boots
// Whiskers
// Bounder
```

```
// Snoopy
// Fido
```

Remarks

The Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>) method generates a MethodCallExpression that represents calling Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Concat<TSource>(IQueryable<TSource>, IEnumerable<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that the elements in `source2` are concatenated to those of `source1` to create a new sequence.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Contains Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Determines whether an IQueryable<T> contains a specified element.

Overloads

Expand table

Contains<TSource>(IQueryable<TSource>, TSource)	Determines whether a sequence contains a specified element by using the default equality comparer.
Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>)	Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.

Contains<TSource>(IQueryable<TSource>, TSource)

Source: Queryable.cs

Determines whether a sequence contains a specified element by using the default equality comparer.

C#

```
public static bool Contains<TSource> (this
System.Linq.IQueryable<TSource> source, TSource item);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> in which to locate `item`.

`item` TSource

The object to locate in the sequence.

Returns

Boolean

`true` if the input sequence contains an element that has the specified value; otherwise, `false`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Contains<TSource> (IQueryable<TSource>, TSource) to determine whether a sequence contains a specific element.

C#

```
string[] fruits = { "apple", "banana", "mango",
                    "orange", "passionfruit", "grape" };

// The string to search for in the array.
string mango = "mango";

bool hasMango = fruits.AsQueryable().Contains(mango);

Console.WriteLine(
    "The array {0} contain '{1}'.",
    hasMango ? "does" : "does not",
    mango);

// This code produces the following output:
//
// The array does contain 'mango'.
```

Remarks

The Contains<TSource>(IQueryable<TSource>, TSource) method generates a MethodCallExpression that represents calling Contains<TSource>(IQueryable<TSource>, TSource) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Contains<TSource>(IQueryable<TSource>, TSource) depends on the implementation of the type of the `source` parameter. The expected behavior is that it determines if `source` contains `item`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>)

Source: Queryable.cs

Determines whether a sequence contains a specified element by using a specified IEqualityComparer<T>.

C#

```
public static bool Contains<TSource> (this
System.Linq.IQueryable<TSource> source, TSource item,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> in which to locate `item`.

`item` TSource

The object to locate in the sequence.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

Boolean

`true` if the input sequence contains an element that has the specified value; otherwise, `false`.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>) method generates a MethodCallExpression that represents calling Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Contains<TSource>(IQueryable<TSource>, TSource, IEqualityComparer<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it determines if `source` contains `item` by using `comparer` to compare values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Count Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns the number of elements in a sequence.

Overloads

Expand table

Count<TSource>(IQueryable<TSource>)	Returns the number of elements in a sequence.
Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the number of elements in the specified sequence that satisfies a condition.

Count<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the number of elements in a sequence.

C#

```
public static int Count<TSource> (this System.Linq.IQueryable<TSource>
source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

The IQueryable<T> that contains the elements to be counted.

Returns

Int32

The number of elements in the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The number of elements in `source` is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Count<TSource>(IQueryable<TSource>) to count the elements in a sequence.

C#

```
string[] fruits = { "apple", "banana", "mango",
                    "orange", "passionfruit", "grape" };

int numberOfFruits = fruits.AsQueryable().Count();

Console.WriteLine(
    "There are {0} items in the array.",
    numberOfFruits);

// This code produces the following output:
//
// There are 6 items in the array.
```

Remarks

The Count<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Count<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Count<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it counts the number of items in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the number of elements in the specified sequence that satisfies a condition.

C#

```
public static int Count<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,bool>>
predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> that contains the elements to be counted.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

Int32

The number of elements in the sequence that satisfies the condition in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

OverflowException

The number of elements in `source` is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Count<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to count the elements in a sequence that satisfy a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public bool Vaccinated { get; set; }
}

public static void CountEx2()
{
    // Create an array of Pet objects.
    Pet[] pets = { new Pet { Name="Barley", Vaccinated=true },
                   new Pet { Name="Boots", Vaccinated=false },
                   new Pet { Name="Whiskers", Vaccinated=false } };

    // Count the number of unvaccinated pets in the array.
    int numberUnvaccinated =
        pets.AsQueryable().Count(p => p.Vaccinated == false);

    Console.WriteLine(
        "There are {0} unvaccinated animals.",
        numberUnvaccinated);
}
```

```
// This code produces the following output:
//
// There are 2 unvaccinated animals.
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Count<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it counts the number of items in `source` that satisfy the condition specified by `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.DefaultIfEmpty Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns the elements in a sequence or a default valued singleton collection if the sequence is empty.

Overloads

Expand table

DefaultIfEmpty<TSource> (IQueryable<TSource>, TSource)	Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.
DefaultIfEmpty<TSource> (IQueryable<TSource>)	Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.

DefaultIfEmpty<TSource> (IQueryable<TSource>, TSource)

Source: Queryable.cs

Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.

C#

```
public static System.Linq.IQueryable<TSource> DefaultIfEmpty<TSource>
(this System.Linq.IQueryable<TSource> source, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The IQueryable<T> to return the specified value for if empty.

`defaultValue` TSource

The value to return if the sequence is empty.

Returns

IQueryable<TSource>

An IQueryable<T> that contains `defaultValue` if `source` is empty; otherwise, `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example shows a situation in which it is useful to call DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) in a LINQ query. A default value is passed to DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) in this example.

C#

```
// Create a list of Pet objects.
List<Pet> pets =
    new List<Pet>{ new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

// This query selects only those pets that are 10 or older.
// In case there are no pets that meet that criteria, call
// DefaultIfEmpty(). This code passes an (optional) default
// value to DefaultIfEmpty().
string[] oldPets =
    pets.AsQueryable()
    .Where(pet => pet.Age >= 10)
    .Select(pet => pet.Name)
    .DefaultIfEmpty("[EMPTY]")
    .ToArray();
```

```
Console.WriteLine("First query: " + oldPets[0]);

// This query selects only those pets that are 10 or older.
// This code does not call DefaultIfEmpty().
string[] oldPets2 =
    pets.AsQueryable()
    .Where(pet => pet.Age >= 10)
    .Select(pet => pet.Name)
    .ToArray();

// There may be no elements in the array, so directly
// accessing element 0 may throw an exception.
try
{
    Console.WriteLine("Second query: " + oldPets2[0]);
}
catch (Exception e)
{
    Console.WriteLine("Second query: An exception was thrown: " +
e.Message);
}

/*
    This code produces the following output:

    First query: [EMPTY]
    Second query: An exception was thrown: Index was outside the bounds
of the array.
*/
```

Remarks

The DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method generates a MethodCallExpression that represents calling DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns `source` if it is not empty. Otherwise, it returns an IQueryable<T> that contains `defaultValue`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

DefaultIfEmpty<TSource> (IQueryable<TSource>)

Source: Queryable.cs

Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.

C#

```
public static System.Linq.IQueryable<TSource?> DefaultIfEmpty<TSource>
(this System.Linq.IQueryable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The IQueryable<T> to return a default value for if empty.

Returns

IQueryable<TSource>

An IQueryable<T> that contains `default`(`TSource`) if `source` is empty; otherwise, `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code examples demonstrate how to use DefaultIfEmpty<TSource> (IQueryable<TSource>) to provide a default value in case the source sequence is empty.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void DefaultIfEmptyEx1()
{
    // Create a list of Pet objects.
    List<Pet> pets =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 } };

    // Call DefaultIfEmtpy() on the collection that Select()
    // returns, so that if the initial list is empty, there
    // will always be at least one item in the returned array.
    string[] names =
        pets.AsQueryable()
        .Select(pet => pet.Name)
        .DefaultIfEmpty()
        .ToArray();

    string first = names[0];
    Console.WriteLine(first);
}

/*
    This code produces the following output:

    Barley
*/
```

Remarks

The DefaultIfEmpty<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling DefaultIfEmpty<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling DefaultIfEmpty<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns `source` if it is not empty. Otherwise, it returns an IQueryable<T> that contains `default(TSource)`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Distinct Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns distinct elements from a sequence.

Overloads

Expand table

Distinct<TSource>(IQueryable<TSource>)	Returns distinct elements from a sequence by using the default equality comparer to compare values.
Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>)	Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.

Distinct<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns distinct elements from a sequence by using the default equality comparer to compare values.

C#

```
public static System.Linq.IQueryable<TSource> Distinct<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The IQueryable<T> to remove duplicates from.

Returns

IQueryable<TSource>

An IQueryable<T> that contains distinct elements from `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Distinct<TSource>(IQueryable<TSource>) to return distinct elements from a sequence.

C#

```
List<int> ages = new List<int> { 21, 46, 46, 55, 17, 21, 55, 55 };

IEnumerable<int> distinctAges = ages.AsQueryable().Distinct();

Console.WriteLine("Distinct ages:");

foreach (int age in distinctAges)
    Console.WriteLine(age);

/*
    This code produces the following output:

    Distinct ages:
    21
    46
    55
    17
*/
```

Remarks

The Distinct<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Distinct<TSource>

(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Distinct<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns an unordered sequence of the unique items in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>)

Source: Queryable.cs

Returns distinct elements from a sequence by using a specified IEqualityComparer<T> to compare values.

C#

```
public static System.Linq.IQueryable<TSource> Distinct<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The IQueryable<T> to remove duplicates from.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IQueryable<TSource>

An IQueryable<T> that contains distinct elements from `source`.

Exceptions

ArgumentNullException

`source` or `comparer` is `null`.

Remarks

The Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>) method generates a MethodCallExpression that represents calling Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Distinct<TSource>(IQueryable<TSource>, IEqualityComparer<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns an unordered sequence of the unique items in `source` by using `comparer` to compare values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.DistinctBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns distinct elements from a sequence according to a specified key selector function.
DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Returns distinct elements from a sequence according to a specified key selector function.

DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Returns distinct elements from a sequence according to a specified key selector function.

C#

```
public static System.Linq.IQueryable<TSource> DistinctBy<TSource,TKey>
(this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

`TKey`

The type of key to distinguish elements by.

Parameters

`source` IQueryable<TSource>

The sequence to remove duplicate elements from.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

IQueryable<TSource>

An IQueryable<T> that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

DistinctBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Returns distinct elements from a sequence according to a specified key selector function.

```C#
public static System.Linq.IQueryable<TSource> DistinctBy<TSource,TKey>
(this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of key to distinguish elements by.

Parameters

`source` IQueryable<TSource>

The sequence to remove duplicate elements from.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<TSource>

An IQueryable<T> that contains distinct elements from the source sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.ElementAt Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

ElementAt<TSource>(IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence.
ElementAt<TSource>(IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence.

ElementAt<TSource>(IQueryable<TSource>, Index)

Source: Queryable.cs

Returns the element at a specified index in a sequence.

C#

```
public static TSource ElementAt<TSource> (this
System.Linq.IQueryable<TSource> source, Index index);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

An IQueryable<T> to return an element from.

`index` Index

The index of the element to retrieve, which is either from the start or the end.

Returns

TSource

The element at the specified position in the `source` sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentOutOfRangeException

`index` is outside the bounds of the `source` sequence.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ElementAt<TSource>(IQueryable<TSource>, Int32)

Source: Queryable.cs

Returns the element at a specified index in a sequence.

```
C#

public static TSource ElementAt<TSource> (this
System.Linq.IQueryable<TSource> source, int index);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`index` Int32

The zero-based index of the element to retrieve.

Returns

TSource

The element at the specified position in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentOutOfRangeException

`index` is less than zero.

Examples

The following code example demonstrates how to use ElementAt<TSource> (IQueryable<TSource>, Int32) to return an element at a specific position in a sequence.

C#

```
string[] names = { "Hartono, Tommy", "Adams, Terry",
                   "Andersen, Henriette Thaulow",
                   "Hedlund, Magnus", "Ito, Shu" };

Random random = new Random(DateTime.Now.Millisecond);

string name =
    names.AsQueryable().ElementAt(random.Next(0, names.Length));

Console.WriteLine("The name chosen at random is '{0}'.", name);

/*
    This code produces the following sample output.
    Yours may be different due to the use of Random.
```

```
    The name chosen at random is 'Ito, Shu'.
*/
```

Remarks

The ElementAt<TSource>(IQueryable<TSource>, Int32) method generates a MethodCallExpression that represents calling ElementAt<TSource>(IQueryable<TSource>, Int32) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling ElementAt<TSource>(IQueryable<TSource>, Int32) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the item at position `index` in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.ElementAtOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

ElementAtOrDefault<TSource> (IQueryable<TSource>, Index)	Returns the element at a specified index in a sequence or a default value if the index is out of range.
ElementAtOrDefault<TSource> (IQueryable<TSource>, Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.

ElementAtOrDefault<TSource> (IQueryable<TSource>, Index)

Source: Queryable.cs

Returns the element at a specified index in a sequence or a default value if the index is out of range.

C#

```
public static TSource? ElementAtOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source, Index index);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`index` Index

The index of the element to retrieve, which is either from the start or the end.

Returns

TSource

`default` if `index` is outside the bounds of the `source` sequence; otherwise, the element at the specified position in the `source` sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ElementAtOrDefault<TSource> (IQueryable<TSource>, Int32)

Source: Queryable.cs

Returns the element at a specified index in a sequence or a default value if the index is out of range.

C#

```
public static TSource? ElementAtOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source, int index);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`index` Int32

The zero-based index of the element to retrieve.

Returns

TSource

`default`(`TSource`) if `index` is outside the bounds of `source`; otherwise, the element at the specified position in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32). This example uses a value for `index` that is outside the bounds of the source sequence.

C#

```
string[] names = { "Hartono, Tommy", "Adams, Terry",
                   "Andersen, Henriette Thaulow",
                   "Hedlund, Magnus", "Ito, Shu" };

int index = 20;

string name = names.AsQueryable().ElementAtOrDefault(index);

Console.WriteLine(
    "The name chosen at index {0} is '{1}'.",
    index,
    String.IsNullOrEmpty(name) ? "[NONE AT THIS INDEX]" : name);

/*
```

```
    This code produces the following output:

    The name chosen at index 20 is '[NONE AT THIS INDEX]'.
*/
```

Remarks

The ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32) method generates a MethodCallExpression that represents calling ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling ElementAtOrDefault<TSource>(IQueryable<TSource>, Int32) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the item at position `index` in `source`, or `default(TSource)` if `index` is outside the bounds of `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Except Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Produces the set difference of two sequences.

Overloads

Expand table

Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set difference of two sequences by using the default equality comparer to compare values.
Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.

Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>)

Source: Queryable.cs

Produces the set difference of two sequences by using the default equality comparer to compare values.

C#

```
public static System.Linq.IQueryable<TSource> Except<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose elements that are not also in `source2` will be returned.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose elements that also occur in the first sequence will not appear in the returned sequence.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the set difference of the two sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Examples

The following code example demonstrates how to use Except<TSource> (IQueryable<TSource>, IEnumerable<TSource>) to return those elements that only appear in the first source sequence.

C#

```
double[] numbers1 = { 2.0, 2.1, 2.2, 2.3, 2.4, 2.5 };
double[] numbers2 = { 2.2 };

// Get the numbers from the first array that
// are NOT in the second array.
IEnumerable<double> onlyInFirstSet =
    numbers1.AsQueryable().Except(numbers2);

foreach (double number in onlyInFirstSet)
    Console.WriteLine(number);

/*
    This code produces the following output:

    2
    2.1
    2.3
    2.4
```

```
    2.5
*/
```

Remarks

The Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>) method generates a MethodCallExpression that represents calling Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that all the elements in `source1` are returned except for those that are also in `source2`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Queryable.cs

Produces the set difference of two sequences by using the specified IEqualityComparer<T> to compare values.

```
C#

public static System.Linq.IQueryable<TSource> Except<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose elements that are not also in `source2` will be returned.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose elements that also occur in the first sequence will not appear in the returned sequence.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the set difference of the two sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Remarks

The Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method generates a MethodCallExpression that

represents calling Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Except<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that all the elements in `source1` are returned except for those that are also in `source2`, and `comparer` is used to compare values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.ExceptBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set difference of two sequences according to a specified key selector function.
ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set difference of two sequences according to a specified key selector function.

ExceptBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Produces the set difference of two sequences according to a specified key selector function.

C#

```
public static System.Linq.IQueryable<TSource> ExceptBy<TSource,TKey>
(this System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TKey> source2,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequence.

`TKey`

The type of key to identify elements by.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose keys that are not also in `source2` will be returned.

`source2` IEnumerable<TKey>

An IEnumerable<T> whose keys that also occur in the first sequence will cause those elements to be removed from the returned sequence.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<TSource>

A IQueryable<T> that contains the set difference of the elements of two sequences.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

ExceptBy<TSource,TKey> (IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Produces the set difference of two sequences according to a specified key selector function.

```
C#

public static System.Linq.IQueryable<TSource> ExceptBy<TSource,TKey>
(this System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TKey> source2,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

`TSource`

The type of the elements of the input sequence.

`TKey`

The type of key to identify elements by.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose keys that are not also in `source2` will be returned.

`source2` IEnumerable<TKey>

An IEnumerable<T> whose keys that also occur in the first sequence will cause those elements to be removed from the returned sequence.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

IQueryable<TSource>

A IQueryable<T> that contains the set difference of the elements of two sequences.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.First Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns the first element of a sequence.

Overloads

Expand table

First<TSource>(IQueryable<TSource>)	Returns the first element of a sequence.
First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition.

First<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the first element of a sequence.

C#

```
public static TSource First<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

The IQueryable<T> to return the first element of.

Returns

TSource

The first element in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The source sequence is empty.

Examples

The following code example demonstrates how to use First<TSource>(IQueryable<TSource>) to return the first element in a sequence.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                    83, 23, 87, 435, 67, 12, 19 };

int first = numbers.AsQueryable().First();

Console.WriteLine(first);

/*
    This code produces the following output:

    9
*/
```

Remarks

The First<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling First<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling First<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the first element in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the first element of a sequence that satisfies a specified condition.

C#

```
public static TSource First<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

TSource

The first element in `source` that passes the test in `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

No element satisfies the condition in `predicate`.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to return the first element of a sequence that satisfies a condition.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                  83, 23, 87, 435, 67, 12, 19 };

// Get the first number in the array that is greater than 80.
int first = numbers.AsQueryable().First(number => number > 80);

Console.WriteLine(first);

/*
    This code produces the following output:

    92
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling First<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the first element in `source` that satisfies the condition specified by `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.FirstOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns the first element of a sequence, or a default value if no element is found.

Overloads

Expand table

FirstOrDefault<TSource>(IQueryable<TSource>)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.
FirstOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the first element of a sequence, or a default value if the sequence contains no elements.
FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.

FirstOrDefault<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the first element of a sequence, or a default value if the sequence contains no elements.

```
C#
```

```
public static TSource? FirstOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The IQueryable<T> to return the first element of.

Returns

TSource

`default`(`TSource`) if `source` is empty; otherwise, the first element in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use FirstOrDefault<TSource> (IQueryable<TSource>) on an empty sequence.

C#

```
// Create an empty array.
int[] numbers = { };
// Get the first item in the array, or else the
// default value for type int (0).
int first = numbers.AsQueryable().FirstOrDefault();

Console.WriteLine(first);

/*
    This code produces the following output:

    0
*/
```

Sometimes the value of `default(TSource)` is not the default value that you want to use if the collection contains no elements. Instead of checking the result for the unwanted default value and then changing it if necessary, you can use the DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method to specify the default value that you want to use if the collection is empty. Then, call First<TSource>(IQueryable<TSource>) to obtain the first element. The following code example uses both techniques to obtain a default value of 1 if a collection of numeric months is empty. Because the default value for an integer is 0, which does not correspond to any month, the default value must be specified as 1 instead. The first result variable is checked for the unwanted default value after the query is completed. The second result variable is obtained by calling DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) to specify a default value of 1.

C#

```
List<int> months = new List<int> { };

// Setting the default value to 1 after the query.
int firstMonth1 = months.AsQueryable().FirstOrDefault();
if (firstMonth1 == 0)
{
    firstMonth1 = 1;
}
Console.WriteLine("The value of the firstMonth1 variable is {0}",
firstMonth1);

// Setting the default value to 1 by using DefaultIfEmpty() in the query.
int firstMonth2 = months.AsQueryable().DefaultIfEmpty(1).First();
Console.WriteLine("The value of the firstMonth2 variable is {0}",
firstMonth2);

/*
 This code produces the following output:

 The value of the firstMonth1 variable is 1
 The value of the firstMonth2 variable is 1
*/
```

Remarks

The FirstOrDefault<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling FirstOrDefault<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling FirstOrDefault<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the first element in `source`, or a default value if `source` is empty.

The FirstOrDefault method does not provide a way to specify the default value to return if `source` is empty. If you want to specify a default value other than `default(TSource)`, use the DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method as described in the Example section.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

FirstOrDefault<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.

C#

```
public static TSource? FirstOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

TSource

`default`(`TSource`) if `source` is empty or if no element passes the test specified by `predicate`; otherwise, the first element in `source` that passes the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) by passing in a predicate. In the second query, there is no element in the sequence that satisfies the condition.

C#

```
string[] names = { "Hartono, Tommy", "Adams, Terry",
                     "Andersen, Henriette Thaulow",
                     "Hedlund, Magnus", "Ito, Shu" };

// Get the first string in the array that is longer
// than 20 characters, or the default value for type
// string (null) if none exists.
string firstLongName =
    names.AsQueryable().FirstOrDefault(name => name.Length > 20);

Console.WriteLine("The first long name is '{0}'.", firstLongName);
```

```
// Get the first string in the array that is longer
// than 30 characters, or the default value for type
// string (null) if none exists.
string firstVeryLongName =
    names.AsQueryable().FirstOrDefault(name => name.Length > 30);

Console.WriteLine(
    "There is {0} name that is longer than 30 characters.",
    string.IsNullOrEmpty(firstVeryLongName) ? "NOT a" : "a");

/*
    This code produces the following output:

    The first long name is 'Andersen, Henriette Thaulow'.
    There is NOT a name that is longer than 30 characters.
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling FirstOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the first element in `source` that satisfies the condition in `predicate`, or a default value if no element satisfies the condition.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

FirstOrDefault<TSource> (IQueryable<TSource>, TSource)

Source: Queryable.cs

Returns the first element of a sequence, or a default value if the sequence contains no elements.

```
C#

public static TSource FirstOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The IEnumerable<T> to return the first element of.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if `source` is empty; otherwise, the first element in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

FirstOrDefault<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)

Source: Queryable.cs

Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.

```
C#

public static TSource FirstOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if `source` is empty or if no element passes the test specified by `predicate`; otherwise, the first element in `source` that passes the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.GroupBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Groups the elements of a sequence.

Overloads

Expand table

GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.
GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>)	Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>)	Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.
GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Groups the elements of a sequence according to a specified key selector function.
GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey,

IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.

C#

```
public static System.Linq.IQueryable<TResult>
GroupBy<TSource,TKey,TElement,TResult> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Linq.Expressions.Expression<Func<TSource,TElement>>
elementSelector,
System.Linq.Expressions.Expression<Func<TKey,System.Collections.Generic.I
Enumerable<TElement>,TResult>> resultSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

`TElement`

The type of the elements in each IGrouping<TKey,TElement>.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

elementSelector Expression<Func<TSource,TElement>>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

resultSelector Expression<Func<TKey,IEnumerable<TElement>,TResult>>

A function to create a result value from each group.

comparer IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<TResult>

An `T:System.Linq.IQueryable`1` that has a type argument of `TResult` and where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` or `resultSelector` or `comparer` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>, IEqualityComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by key values that are obtained by invoking `keySelector` on each element. The `comparer` parameter is used to compare key values. The `elementSelector` parameter is used to project the elements of each group, and the `resultSelector` parameter is used to obtain a result value from each group and its key.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.

C#

```
public static System.Linq.IQueryable<TResult>
GroupBy<TSource,TKey,TElement,TResult> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Linq.Expressions.Expression<Func<TSource,TElement>>
elementSelector,
System.Linq.Expressions.Expression<Func<TKey,System.Collections.Generic.I
Enumerable<TElement>,TResult>> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

`TElement`

The type of the elements in each IGrouping<TKey,TElement>.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`elementSelector` Expression<Func<TSource,TElement>>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

`resultSelector` Expression<Func<TKey,IEnumerable<TElement>,TResult>>

A function to create a result value from each group.

Returns

IQueryable<TResult>

An `T:System.Linq.IQueryable`1` that has a type argument of `TResult` and where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>) to group the elements of a sequence and project a sequence of results of type `TResult`.

C#

```
class Pet
{
    public string Name { get; set; }
    public double Age { get; set; }
}

public static void GroupByEx4()
{
    // Create a list of pets.
    List<Pet> petsList =
        new List<Pet>{ new Pet { Name="Barley", Age=8.3 },
                       new Pet { Name="Boots", Age=4.9 },
                       new Pet { Name="Whiskers", Age=1.5 },
                       new Pet { Name="Daisy", Age=4.3 } };

    // Group Pet.Age values by the Math.Floor of the age.
    // Then project an anonymous type from each group
    // that consists of the key, the count of the group's
    // elements, and the minimum and maximum age in the group.
    var query = petsList.AsQueryable().GroupBy(
        pet => Math.Floor(pet.Age),
        pet => pet.Age,
        (baseAge, ages) => new
        {
            Key = baseAge,
            Count = ages.Count(),
            Min = ages.Min(),
            Max = ages.Max()
        });
```

```
    // Iterate over each anonymous type.
    foreach (var result in query)
    {
        Console.WriteLine("\nAge group: " + result.Key);
        Console.WriteLine("Number of pets in this age group: " +
result.Count);
        Console.WriteLine("Minimum age: " + result.Min);
        Console.WriteLine("Maximum age: " + result.Max);
    }

    /*  This code produces the following output:

        Age group: 8
        Number of pets in this age group: 1
        Minimum age: 8.3
        Maximum age: 8.3

        Age group: 4
        Number of pets in this age group: 2
        Minimum age: 4.3
        Maximum age: 4.9

        Age group: 1
        Number of pets in this age group: 1
        Minimum age: 1.5
        Maximum age: 1.5
    */
}
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey,TElement,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TSource,TElement>>, Expression<Func<TKey, IEnumerable<TElement>,TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by key values that are obtained by invoking `keySelector` on each element. The `elementSelector` parameter is used to project the elements of each group, and the `resultSelector` parameter is used to obtain a result value from each group and its key.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.

C#

```
public static
System.Linq.IQueryable<System.Linq.IGrouping<TKey,TElement>>
GroupBy<TSource,TKey,TElement> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,TKey>>
```

```
keySelector, System.Linq.Expressions.Expression<Func<TSource,TElement>>
elementSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

`TElement`

The type of the elements in each IGrouping<TKey,TElement>.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`elementSelector` Expression<Func<TSource,TElement>>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

Returns

IQueryable<IGrouping<TKey,TElement>>

An `IQueryable<IGrouping<TKey, TElement>>` in C# or `IQueryable(Of IGrouping(Of TKey, TElement))` in Visual Basic where each IGrouping<TKey,TElement> contains a sequence of objects of type `TElement` and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>) to group the elements of a sequence.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void GroupByEx2()
{
    // Create a list of Pet objects.
    List<Pet> pets =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 },
                       new Pet { Name="Daisy", Age=4 } };

    // Group the pets using Pet.Age as the key.
    // Use Pet.Name as the value for each entry.
    IEnumerable<IGrouping<int, string>> query =
        pets.AsQueryable().GroupBy(pet => pet.Age, pet => pet.Name);

    // Iterate over each IGrouping in the collection.
    foreach (IGrouping<int, string> petGroup in query)
    {
        // Print the key value of the IGrouping.
        Console.WriteLine(petGroup.Key);
        // Iterate over each value in the
        // IGrouping and print the value.
        foreach (string name in petGroup)
            Console.WriteLine("  {0}", name);
    }
}

/*
    This code produces the following output:

    8
      Barley
    4
      Boots
      Daisy
    1
      Whiskers
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by a key value that is obtained by invoking `keySelector` on each element. It invokes `elementSelector` on each element to obtain a result element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>,

Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.

C#

```
public static
System.Linq.IQueryable<System.Linq.IGrouping<TKey,TElement>>
GroupBy<TSource,TKey,TElement> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,TKey>>
keySelector, System.Linq.Expressions.Expression<Func<TSource,TElement>>
elementSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

`TElement`

The type of the elements in each IGrouping<TKey,TElement>.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`elementSelector` Expression<Func<TSource,TElement>>

A function to map each source element to an element in an IGrouping<TKey,TElement>.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<IGrouping<TKey,TElement>>

An `IQueryable<IGrouping<TKey, TElement>>` in C# or `IQueryable(Of IGrouping(Of TKey, TElement))` in Visual Basic where each IGrouping<TKey,TElement> contains a sequence of objects of type `TElement` and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `elementSelector` or `comparer` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey,TElement>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TSource,TElement>>, IEqualityComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by a key value that is obtained by invoking `keySelector` on each element. Key values are compared by using `comparer`. The `elementSelector` parameter is invoked on each element to obtain a result element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TResult> (IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.

C#

```
public static System.Linq.IQueryable<TResult>
GroupBy<TSource,TKey,TResult> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,TKey>>
keySelector,
System.Linq.Expressions.Expression<Func<TKey,System.Collections.Generic.I
Enumerable<TSource>,TResult>> resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`resultSelector` Expression<Func<TKey,IEnumerable<TSource>,TResult>>

A function to create a result value from each group.

Returns

IQueryable<TResult>

An `T:System.Linq.IQueryable`1` that has a type argument of `TResult` and where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>) to group the elements of a sequence and project a sequence of results of type `TResult`.

C#

```
class Pet
{
    public string Name { get; set; }
    public double Age { get; set; }
}

public static void GroupByEx3()
```

```
{
    // Create a list of pets.
    List<Pet> petsList =
        new List<Pet>{ new Pet { Name="Barley", Age=8.3 },
                       new Pet { Name="Boots", Age=4.9 },
                       new Pet { Name="Whiskers", Age=1.5 },
                       new Pet { Name="Daisy", Age=4.3 } };

    // Group Pet objects by the Math.Floor of their age.
    // Then project an anonymous type from each group
    // that consists of the key, the count of the group's
    // elements, and the minimum and maximum age in the group.
    var query = petsList.AsQueryable().GroupBy(
        pet => Math.Floor(pet.Age),
        (age, pets) => new
        {
            Key = age,
            Count = pets.Count(),
            Min = pets.Min(pet => pet.Age),
            Max = pets.Max(pet => pet.Age)
        });

    // Iterate over each anonymous type.
    foreach (var result in query)
    {
        Console.WriteLine("\nAge group: " + result.Key);
        Console.WriteLine("Number of pets in this age group: " +
result.Count);
        Console.WriteLine("Minimum age: " + result.Min);
        Console.WriteLine("Maximum age: " + result.Max);
    }

    /*  This code produces the following output:

        Age group: 8
        Number of pets in this age group: 1
        Minimum age: 8.3
        Maximum age: 8.3

        Age group: 4
        Number of pets in this age group: 2
        Minimum age: 4.3
        Maximum age: 4.9

        Age group: 1
        Number of pets in this age group: 1
        Minimum age: 1.5
        Maximum age: 1.5
    */
}
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by a key value that is obtained by invoking `keySelector` on each element. The `resultSelector` parameter is used to obtain a result value from each group and its key.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>,

Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer.

```C#
public static System.Linq.IQueryable<TResult>
GroupBy<TSource,TKey,TResult> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,TKey>>
keySelector,
System.Linq.Expressions.Expression<Func<TKey,System.Collections.Generic.I
Enumerable<TSource>,TResult>> resultSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

`TResult`

The type of the result value returned by `resultSelector`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`resultSelector` Expression<Func<TKey,IEnumerable<TSource>,TResult>>

A function to create a result value from each group.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<TResult>

An `T:System.Linq.IQueryable`1` that has a type argument of `TResult` and where each element represents a projection over a group and its key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `resultSelector` or `comparer` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEqualityComparer<TKey>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEqualityComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey,TResult>(IQueryable<TSource>, Expression<Func<TSource, TKey>>, Expression<Func<TKey,IEnumerable<TSource>,TResult>>, IEqualityComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by key values that are obtained by invoking `keySelector` on each element. The `comparer` parameter is used to compare keys and the `resultSelector` parameter is used to obtain a result value from each group and its key.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupBy<TSource,TKey> (IQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function.

C#

```
public static System.Linq.IQueryable<System.Linq.IGrouping<TKey,TSource>>
GroupBy<TSource,TKey> (this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

keySelector Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

IQueryable<IGrouping<TKey,TSource>>

An `IQueryable<IGrouping<TKey, TSource>>` in C# or `IQueryable(Of IGrouping(Of TKey, TSource))` in Visual Basic where each IGrouping<TKey,TElement> object contains a sequence of objects and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) to group the elements of a sequence.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void GroupByEx1()
{
    // Create a list of Pet objects.
    List<Pet> pets =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 },
                       new Pet { Name="Daisy", Age=4 } };

    // Group the pets using Pet.Age as the key.
    // Use Pet.Name as the value for each entry.
    var query = pets.AsQueryable().GroupBy(pet => pet.Age);

    // Iterate over each IGrouping in the collection.
    foreach (var ageGroup in query)
    {
        Console.WriteLine("Age group: {0}  Number of pets: {1}",
ageGroup.Key, ageGroup.Count());
```

```
    }
}

/*
    This code produces the following output:

    Age group: 8  Number of pets: 1
    Age group: 4  Number of pets: 2
    Age group: 1  Number of pets: 1

*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by a key value that is obtained by invoking `keySelector` on each element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

GroupBy<TSource,TKey> (IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.

C#

```
public static System.Linq.IQueryable<System.Linq.IGrouping<TKey,TSource>>
GroupBy<TSource,TKey> (this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented in `keySelector`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> whose elements to group.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<IGrouping<TKey,TSource>>

An `IQueryable<IGrouping<TKey, TSource>>` in C# or `IQueryable(Of IGrouping(Of TKey, TSource))` in Visual Basic where each IGrouping<TKey,TElement> contains a sequence of objects and a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `comparer` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>) method generates a MethodCallExpression that represents calling GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it groups the elements of `source` by a key value. The key value is obtained by invoking `keySelector` on each element, and key values are compared by using `comparer`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7,

Product	Versions
	8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.GroupJoin Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Correlates the elements of two sequences based on key equality and groups the results.

Overloads

Expand table

GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>)	Correlates the elements of two sequences based on key equality and groups the results. The default equality comparer is used to compare keys.
GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.

GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>)

Source: Queryable.cs

Correlates the elements of two sequences based on key equality and groups the results. The default equality comparer is used to compare keys.

C#

```
public static System.Linq.IQueryable<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
System.Linq.IQueryable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner,
System.Linq.Expressions.Expression<Func<TOuter,TKey>> outerKeySelector,
System.Linq.Expressions.Expression<Func<TInner,TKey>> innerKeySelector,
System.Linq.Expressions.Expression<Func<TOuter,System.Collections.Generic
.IEnumerable<TInner>,TResult>> resultSelector);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IQueryable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Expression<Func<TOuter,TKey>>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Expression<Func<TInner,TKey>>

A function to extract the join key from each element of the second sequence.

`resultSelector` Expression<Func<TOuter,IEnumerable<TInner>,TResult>>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

Returns

IQueryable<TResult>

An IQueryable<T> that contains elements of type `TResult` obtained by performing a grouped join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>) to perform a grouped join on two sequences.

C#

```
class Person
{
    public string Name { get; set; }
}

class Pet
{
    public string Name { get; set; }
    public Person Owner { get; set; }
}

public static void GroupJoinEx1()
{
    Person magnus = new Person { Name = "Hedlund, Magnus" };
    Person terry = new Person { Name = "Adams, Terry" };
    Person charlotte = new Person { Name = "Weiss, Charlotte" };

    Pet barley = new Pet { Name = "Barley", Owner = terry };
    Pet boots = new Pet { Name = "Boots", Owner = terry };
    Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };
    Pet daisy = new Pet { Name = "Daisy", Owner = magnus };
```

```
    List<Person> people = new List<Person> { magnus, terry, charlotte };
    List<Pet> pets = new List<Pet> { barley, boots, whiskers, daisy };

    // Create a list where each element is an anonymous
    // type that contains a person's name and a collection
    // of names of the pets that are owned by them.
    var query =
        people.AsQueryable().GroupJoin(pets,
                         person => person,
                         pet => pet.Owner,
                         (person, petCollection) =>
                             new
                             {
                                 OwnerName = person.Name,
                                 Pets = petCollection.Select(pet =>
pet.Name)
                             });

    foreach (var obj in query)
    {
        // Output the owner's name.
        Console.WriteLine("{0}:", obj.OwnerName);
        // Output each of the owner's pet's names.
        foreach (string pet in obj.Pets)
            Console.WriteLine("  {0}", pet);
    }
}

/*
    This code produces the following output:

    Hedlund, Magnus:
      Daisy
    Adams, Terry:
      Barley
      Boots
    Weiss, Charlotte:
      Whiskers
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>,

Expression<Func<TOuter,IEnumerable<TInner>,TResult>>) method generates a MethodCallExpression that represents calling GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `outer` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>) depends on the implementation of the type of the `outer` parameter. The expected behavior is that the `outerKeySelector` and `innerKeySelector` functions are used to extract keys from `outer` and `inner`, respectively. These keys are compared for equality to match each element in `outer` with zero or more elements from `inner`. Then the `resultSelector` function is invoked to project a result object from each group of correlated elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>,

Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer<T> is used to compare keys.

C#

```
public static System.Linq.IQueryable<TResult>
GroupJoin<TOuter,TInner,TKey,TResult> (this
System.Linq.IQueryable<TOuter> outer,
System.Collections.Generic.IEnumerable<TInner> inner,
System.Linq.Expressions.Expression<Func<TOuter,TKey>> outerKeySelector,
System.Linq.Expressions.Expression<Func<TInner,TKey>> innerKeySelector,
System.Linq.Expressions.Expression<Func<TOuter,System.Collections.Generic
.IEnumerable<TInner>,TResult>> resultSelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IQueryable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Expression<Func<TOuter,TKey>>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Expression<Func<TInner,TKey>>

A function to extract the join key from each element of the second sequence.

`resultSelector` Expression<Func<TOuter,IEnumerable<TInner>,TResult>>

A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.

`comparer` IEqualityComparer<TKey>

A comparer to hash and compare keys.

Returns

IQueryable<TResult>

An IQueryable<T> that contains elements of type `TResult` obtained by performing a grouped join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>) method generates a MethodCallExpression that represents calling GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>) itself as a constructed generic method. It then passes the

MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `outer` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling GroupJoin<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,IEnumerable<TInner>,TResult>>, IEqualityComparer<TKey>) depends on the implementation of the type of the `outer` parameter. The expected behavior is that the `outerKeySelector` and `innerKeySelector` functions are used to extract keys from `outer` and `inner`, respectively. These keys are compared for equality by using `comparer`. The outcome of the comparisons is used to match each element in `outer` with zero or more elements from `inner`. Then the `resultSelector` function is invoked to project a result object from each group of correlated elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Intersect Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Produces the set intersection of two sequences.

Overloads

Expand table

Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set intersection of two sequences by using the default equality comparer to compare values.
Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>)

Source: Queryable.cs

Produces the set intersection of two sequences by using the default equality comparer to compare values.

C#

```
public static System.Linq.IQueryable<TSource> Intersect<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

A sequence whose distinct elements that also appear in `source2` are returned.

`source2` IEnumerable<TSource>

A sequence whose distinct elements that also appear in the first sequence are returned.

Returns

IQueryable<TSource>

A sequence that contains the set intersection of the two sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Examples

The following code example demonstrates how to use Intersect<TSource> (IQueryable<TSource>, IEnumerable<TSource>) to return the elements that appear in each of two sequences.

C#

```
int[] id1 = { 44, 26, 92, 30, 71, 38 };
int[] id2 = { 39, 59, 83, 47, 26, 4, 30 };

// Get the numbers that occur in both arrays (id1 and id2).
IEnumerable<int> both = id1.AsQueryable().Intersect(id2);

foreach (int id in both)
    Console.WriteLine(id);

/*
    This code produces the following output:

    26
    30
*/
```

Remarks

The Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>) method generates a MethodCallExpression that represents calling Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that all the elements in `source1` that are also in `source2` are returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Queryable.cs

Produces the set intersection of two sequences by using the specified IEqualityComparer<T> to compare values.

C#

```
public static System.Linq.IQueryable<TSource> Intersect<TSource> (this
System.Linq.IQueryable<TSource> source1,
```

```
System.Collections.Generic.IEnumerable<TSource> source2,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose distinct elements that also appear in `source2` are returned.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements that also appear in the first sequence are returned.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the set intersection of the two sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Remarks

The Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method generates a MethodCallExpression that represents calling Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Intersect<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that all the elements in `source1` that are also in `source2` are returned. The `comparer` parameter is used to compare elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.IntersectBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)	Produces the set intersection of two sequences according to a specified key selector function.
IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set intersection of two sequences according to a specified key selector function.

IntersectBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TKey>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Produces the set intersection of two sequences according to a specified key selector function.

C#

```
public static System.Linq.IQueryable<TSource> IntersectBy<TSource,TKey>
(this System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TKey> source2,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

TSource

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose distinct elements that also appear in `source2` will be returned.

`source2` IEnumerable<TKey>

An IEnumerable<T> whose distinct elements that also appear in the first sequence will be returned.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

IQueryable<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

IntersectBy<TSource,TKey> (IQueryable<TSource>, IEnumerable<TKey>,

Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Produces the set intersection of two sequences according to a specified key selector function.

```
C#

public static System.Linq.IQueryable<TSource> IntersectBy<TSource,TKey>
(this System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TKey> source2,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose distinct elements that also appear in `source2` will be returned.

`source2` IEnumerable<TKey>

An IEnumerable<T> whose distinct elements that also appear in the first sequence will be returned.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to compare keys.

Returns

IQueryable<TSource>

A sequence that contains the elements that form the set intersection of two sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.Join Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Correlates the elements of two sequences based on matching keys.

Overloads

Expand table

Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>)	Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.
Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>)	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>)

Source: Queryable.cs

Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.

C#

```
public static System.Linq.IQueryable<TResult>
Join<TOuter,TInner,TKey,TResult> (this System.Linq.IQueryable<TOuter>
outer, System.Collections.Generic.IEnumerable<TInner> inner,
System.Linq.Expressions.Expression<Func<TOuter,TKey>> outerKeySelector,
System.Linq.Expressions.Expression<Func<TInner,TKey>> innerKeySelector,
System.Linq.Expressions.Expression<Func<TOuter,TInner,TResult>>
resultSelector);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IQueryable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Expression<Func<TOuter,TKey>>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Expression<Func<TInner,TKey>>

A function to extract the join key from each element of the second sequence.

`resultSelector` Expression<Func<TOuter,TInner,TResult>>

A function to create a result element from two matching elements.

Returns

IQueryable<TResult>

An IQueryable<T> that has elements of type `TResult` obtained by performing an inner join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>) to perform an inner join of two sequences based on a common key.

C#

```
class Person
{
    public string Name { get; set; }
}

class Pet
{
    public string Name { get; set; }
    public Person Owner { get; set; }
}

public static void JoinEx1()
{
    Person magnus = new Person { Name = "Hedlund, Magnus" };
    Person terry = new Person { Name = "Adams, Terry" };
    Person charlotte = new Person { Name = "Weiss, Charlotte" };

    Pet barley = new Pet { Name = "Barley", Owner = terry };
    Pet boots = new Pet { Name = "Boots", Owner = terry };
    Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };
    Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

    List<Person> people = new List<Person> { magnus, terry, charlotte };
    List<Pet> pets = new List<Pet> { barley, boots, whiskers, daisy };

    // Join the list of Person objects and the list of Pet objects
    // to create a list of person-pet pairs where each element is
    // an anonymous type that contains the name of pet and the name
    // of the person that owns the pet.
```

```
    var query = people.AsQueryable().Join(pets,
                    person => person,
                    pet => pet.Owner,
                    (person, pet) =>
                        new { OwnerName = person.Name, Pet = pet.Name });

    foreach (var obj in query)
    {
        Console.WriteLine(
            "{0} - {1}",
            obj.OwnerName,
            obj.Pet);
    }
}

/*
    This code produces the following output:

    Hedlund, Magnus - Daisy
    Adams, Terry - Barley
    Adams, Terry - Boots
    Weiss, Charlotte - Whiskers
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>) method generates a MethodCallExpression that represents calling Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `outer` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>) depends on the implementation of the type of the `outer` parameter. The expected behavior is that of an inner join. The `outerKeySelector` and `innerKeySelector`

functions are used to extract keys from `outer` and `inner`, respectively. These keys are compared for equality to match elements from each sequence. A pair of elements is stored for each element in `inner` that matches an element in `outer`. Then the `resultSelector` function is invoked to project a result object from each pair of matching elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Join<TOuter,TInner,TKey,TResult> (IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

C#

```
public static System.Linq.IQueryable<TResult>
Join<TOuter,TInner,TKey,TResult> (this System.Linq.IQueryable<TOuter>
outer, System.Collections.Generic.IEnumerable<TInner> inner,
System.Linq.Expressions.Expression<Func<TOuter,TKey>> outerKeySelector,
System.Linq.Expressions.Expression<Func<TInner,TKey>> innerKeySelector,
System.Linq.Expressions.Expression<Func<TOuter,TInner,TResult>>
```

```
resultSelector, System.Collections.Generic.IEqualityComparer<TKey>?
comparer);
```

Type Parameters

`TOuter`

The type of the elements of the first sequence.

`TInner`

The type of the elements of the second sequence.

`TKey`

The type of the keys returned by the key selector functions.

`TResult`

The type of the result elements.

Parameters

`outer` IQueryable<TOuter>

The first sequence to join.

`inner` IEnumerable<TInner>

The sequence to join to the first sequence.

`outerKeySelector` Expression<Func<TOuter,TKey>>

A function to extract the join key from each element of the first sequence.

`innerKeySelector` Expression<Func<TInner,TKey>>

A function to extract the join key from each element of the second sequence.

`resultSelector` Expression<Func<TOuter,TInner,TResult>>

A function to create a result element from two matching elements.

`comparer` IEqualityComparer<TKey>

An IEqualityComparer<T> to hash and compare keys.

Returns

IQueryable<TResult>

An IQueryable<T> that has elements of type `TResult` obtained by performing an inner join on two sequences.

Exceptions

ArgumentNullException

`outer` or `inner` or `outerKeySelector` or `innerKeySelector` or `resultSelector` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>) method generates a MethodCallExpression that represents calling Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `outer` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Join<TOuter,TInner,TKey,TResult>(IQueryable<TOuter>, IEnumerable<TInner>, Expression<Func<TOuter,TKey>>, Expression<Func<TInner,TKey>>, Expression<Func<TOuter,TInner,TResult>>, IEqualityComparer<TKey>) depends on the implementation of the type of the `outer` parameter. The expected behavior is that of an inner join. The `outerKeySelector` and `innerKeySelector` functions are used to extract keys from `outer` and `inner`, respectively. These keys are compared for equality by using `comparer`. The outcome of the comparisons is used to create a matching pair for each element in `inner` that matches an element in `outer`. Then the `resultSelector` function is invoked to project a result object from each pair of matching elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Last Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns the last element in a sequence.

Overloads

Expand table

Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a specified condition.
Last<TSource>(IQueryable<TSource>)	Returns the last element in a sequence.

Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the last element of a sequence that satisfies a specified condition.

C#

```
public static TSource Last<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,bool>>
predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

TSource

The last element in `source` that passes the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

No element satisfies the condition in `predicate`.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to return the last element of an array that satisfies a condition.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                    83, 23, 87, 67, 12, 19 };

// Get the last number in the array that is greater than 80.
int last = numbers.AsQueryable().Last(num => num > 80);

Console.WriteLine(last);

/*
    This code produces the following output:

    87
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Last<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the last element in `source` that satisfies the condition specified by `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Last<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the last element in a sequence.

```
C#
```

```
public static TSource Last<TSource> (this System.Linq.IQueryable<TSource>
source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return the last element of.

Returns

TSource

The value at the last position in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The source sequence is empty.

Examples

The following code example demonstrates how to use Last<TSource> (IQueryable<TSource>) to return the last element of an array.

C#

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54,
                    83, 23, 87, 67, 12, 19 };

int last = numbers.AsQueryable().Last();

Console.WriteLine(last);

/*
    This code produces the following output:
```

```
    19
*/
```

Remarks

The Last<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Last<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Last<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the last element in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.LastOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns the last element of a sequence, or a default value if no element is found.

Overloads

Expand table

LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IQueryable<TSource>, TSource)	Returns the last element of a sequence, or a default value if the sequence contains no elements.
LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.
LastOrDefault<TSource>(IQueryable<TSource>)	Returns the last element in a sequence, or a default value if the sequence contains no elements.

LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)

Source: Queryable.cs

Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

```C#
public static TSource LastOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IEnumerable<T> to return an element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if the sequence is empty or if no elements pass the test in the predicate function; otherwise, the last element that passes the test in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

LastOrDefault<TSource> (IQueryable<TSource>, TSource)

Source: Queryable.cs

Returns the last element of a sequence, or a default value if the sequence contains no elements.

```
C#

public static TSource LastOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IEnumerable<T> to return the last element of.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

`defaultValue` if the source sequence is empty; otherwise, the last element in the IEnumerable<T>.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

LastOrDefault<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.

C#

```
public static TSource? LastOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return an element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

TSource

`default(TSource)` if `source` is empty or if no elements pass the test in the predicate function; otherwise, the last element of `source` that passes the test in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use LastOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) by passing in a predicate. In the second call to the method, there is no element in the sequence that satisfies the condition.

C#

```
double[] numbers = { 49.6, 52.3, 51.0, 49.4, 50.2, 48.3 };

// Get the last number in the array that rounds to 50.0,
// or else the default value for type double (0.0).
double last50 =
    numbers.AsQueryable().LastOrDefault(n => Math.Round(n) == 50.0);

Console.WriteLine("The last number that rounds to 50 is {0}.", last50);

// Get the last number in the array that rounds to 40.0,
// or else the default value for type double (0.0).
double last40 =
    numbers.AsQueryable().LastOrDefault(n => Math.Round(n) == 40.0);

Console.WriteLine(
    "The last number that rounds to 40 is {0}.",
    last40 == 0.0 ? "[DOES NOT EXIST]" : last40.ToString());

/*
    This code produces the following output:

    The last number that rounds to 50 is 50.2.
    The last number that rounds to 40 is [DOES NOT EXIST].
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The LastOrDefault<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling LastOrDefault<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling LastOrDefault<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the last element in `source` that satisfies the condition specified by `predicate`. It returns a default value if there is no such element in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

LastOrDefault<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the last element in a sequence, or a default value if the sequence contains no elements.

C#

```
public static TSource? LastOrDefault<TSource> (this
```

```
System.Linq.IQueryable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return the last element of.

Returns

TSource

`default(TSource)` if `source` is empty; otherwise, the last element in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use LastOrDefault<TSource> (IQueryable<TSource>) on an empty array.

C#

```
// Create an empty array.
string[] fruits = { };

// Get the last item in the array, or else the default
// value for type string (null).
string last = fruits.AsQueryable().LastOrDefault();

Console.WriteLine(
    String.IsNullOrEmpty(last) ? "[STRING IS NULL OR EMPTY]" : last);

/*
    This code produces the following output:

    [STRING IS NULL OR EMPTY]
*/
```

Sometimes the value of `default(TSource)` is not the default value that you want to use if the collection contains no elements. Instead of checking the result for the unwanted default value and then changing it if necessary, you can use the DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method to specify the default value that you want to use if the collection is empty. Then, call Last<TSource>(IQueryable<TSource>) to obtain the last element. The following code example uses both techniques to obtain a default value of 1 if a collection of numeric days of the month is empty. Because the default value for an integer is 0, which does not correspond to any day of the month, the default value must be specified as 1 instead. The first result variable is checked for the unwanted default value after the query is completed. The second result variable is obtained by calling DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) to specify a default value of 1.

C#

```
List<int> daysOfMonth = new List<int> { };

// Setting the default value to 1 after the query.
int lastDay1 = daysOfMonth.AsQueryable().LastOrDefault();
if (lastDay1 == 0)
{
    lastDay1 = 1;
}
Console.WriteLine("The value of the lastDay1 variable is {0}", lastDay1);

// Setting the default value to 1 by using DefaultIfEmpty() in the query.
int lastDay2 = daysOfMonth.AsQueryable().DefaultIfEmpty(1).Last();
Console.WriteLine("The value of the lastDay2 variable is {0}", lastDay2);

/*
 This code produces the following output:

 The value of the lastDay1 variable is 1
 The value of the lastDay2 variable is 1
*/
```

Remarks

The LastOrDefault<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling LastOrDefault<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling LastOrDefault<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the last element in `source`, or a default value if `source` is empty.

The LastOrDefault method does not provide a way to specify a default value. If you want to specify a default value other than `default(TSource)`, use the DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method as described in the Example section.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.LongCount Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns an Int64 that represents the number of elements in sequence.

Overloads

Expand table

LongCount<TSource>(IQueryable<TSource>)	Returns an Int64 that represents the total number of elements in a sequence.
LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns an Int64 that represents the number of elements in a sequence that satisfy a condition.

LongCount<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns an Int64 that represents the total number of elements in a sequence.

C#

```
public static long LongCount<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

An IQueryable<T> that contains the elements to be counted.

Returns

Int64

The number of elements in `source`.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The number of elements exceeds Int64.MaxValue.

Examples

The following code example demonstrates how to use LongCount<TSource>(IQueryable<TSource>) to count the elements in an array.

C#

```
string[] fruits = { "apple", "banana", "mango",
                    "orange", "passionfruit", "grape" };

long count = fruits.AsQueryable().LongCount();

Console.WriteLine("There are {0} fruits in the collection.", count);

/*
    This code produces the following output:

    There are 6 fruits in the collection.
*/
```

Remarks

The LongCount<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling LongCount<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling LongCount<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it counts the number of items in `source` and returns an Int64.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns an Int64 that represents the number of elements in a sequence that satisfy a condition.

C#

```
public static long LongCount<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> that contains the elements to be counted.

predicate Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

Int64

The number of elements in `source` that satisfy the condition in the predicate function.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

OverflowException

The number of matching elements exceeds Int64.MaxValue.

Examples

The following code example demonstrates how to use LongCount<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to count the elements in an array that satisfy a condition.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void LongCountEx2()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    const int Age = 3;

    // Count the number of Pet objects where Pet.Age is greater than 3.
    long count = pets.AsQueryable().LongCount(pet => pet.Age > Age);

    Console.WriteLine("There are {0} animals over age {1}.", count, Age);
```

```
}

/*
    This code produces the following output:

    There are 2 animals over age 3.
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling LongCount<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it counts the number of items in `source` that satisfy the condition specified by `predicate` and returns an Int64.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Max Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the maximum resulting value.
Max<TSource>(IQueryable<TSource>)	Returns the maximum value in a generic IQueryable<T>.
Max<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T>.

Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)

Source: Queryable.cs

Invokes a projection function on each element of a generic IQueryable<T> and returns the maximum resulting value.

C#

```
public static TResult? Max<TSource,TResult> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TResult>> selector);
```

Type Parameters

TSource

The type of the elements of `source`.

`TResult`

The type of the value returned by the function represented by `selector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the maximum of.

`selector` Expression<Func<TSource,TResult>>

A projection function to apply to each element.

Returns

TResult

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) to determine the maximum value in a sequence of projected values.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MaxEx2()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
```

```
    // Add Pet.Age to the length of Pet.Name
    // to determine the "maximum" Pet object in the array.
    int max =
        pets.AsQueryable().Max(pet => pet.Age + pet.Name.Length);

    Console.WriteLine(
        "The maximum pet age plus name length is {0}.",
        max);
}

/*
    This code produces the following output:

    The maximum pet age plus name length is 14.
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) method generates a MethodCallExpression that represents calling Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Max<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element in `source` and returns the maximum value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
Framework	
.NET Standard	2.0, 2.1
UWP	10.0

Max<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the maximum value in a generic IQueryable<T>.

```
C#

public static TSource? Max<TSource> (this System.Linq.IQueryable<TSource>
source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the maximum of.

Returns

TSource

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Max<TSource>(IQueryable<TSource>) to determine the maximum value in a sequence.

```
C#

List<long> longs = new List<long> { 4294967296L, 466855135L, 81125L };

long max = longs.AsQueryable().Max();

Console.WriteLine("The largest number is {0}.", max);

/*
    This code produces the following output:

    The largest number is 4294967296.
*/
```

Remarks

The Max<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Max<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Max<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the maximum value in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Max<TSource>(IQueryable<TSource>, IComparer<TSource>)

Source: Queryable.cs

Returns the maximum value in a generic IQueryable<T>.

```
C#

public static TSource? Max<TSource> (this System.Linq.IQueryable<TSource>
source, System.Collections.Generic.IComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the maximum value of.

`comparer` IComparer<TSource>

The IComparer<T> to compare values.

Returns

TSource

The maximum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.MaxBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.
MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.

MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.

C#

```
public static TSource? MaxBy<TSource,TKey> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

TKey

The type of key to compare elements by.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the maximum value of.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

TSource

The value with the maximum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

MaxBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)

Source: Queryable.cs

Returns the maximum value in a generic IQueryable<T> according to a specified key selector function.

```C#
public static TSource? MaxBy<TSource,TKey> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of key to compare elements by.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the maximum value of.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IComparer<TSource>

The IComparer<T> to compare keys.

Returns

TSource

The value with the maximum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.Min Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Invokes a projection function on each element of a generic IQueryable<T> and returns the minimum resulting value.
Min<TSource>(IQueryable<TSource>)	Returns the minimum value of a generic IQueryable<T>.
Min<TSource>(IQueryable<TSource>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T>.

Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)

Source: Queryable.cs

Invokes a projection function on each element of a generic IQueryable<T> and returns the minimum resulting value.

C#

```
public static TResult? Min<TSource,TResult> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TResult>> selector);
```

Type Parameters

TSource

The type of the elements of `source`.

`TResult`

The type of the value returned by the function represented by `selector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the minimum of.

`selector` Expression<Func<TSource,TResult>>

A projection function to apply to each element.

Returns

TResult

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) to determine the minimum value in a sequence of projected values.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void MinEx2()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
```

```
    // Get the Pet object that has the smallest Age value.
    int min = pets.AsQueryable().Min(pet => pet.Age);

    Console.WriteLine("The youngest animal is age {0}.", min);
}

/*
    This code produces the following output:

    The youngest animal is age 1.
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) method generates a MethodCallExpression that represents calling Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Min<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element in `source` and returns the minimum value.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Min<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the minimum value of a generic IQueryable<T>.

```
C#

public static TSource? Min<TSource> (this System.Linq.IQueryable<TSource>
source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the minimum of.

Returns

TSource

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` contains no elements.

Examples

The following code example demonstrates how to use Min<TSource>(IQueryable<TSource>) to determine the minimum value in a sequence.

C#

```
double[] doubles = { 1.5E+104, 9E+103, -2E+103 };

double min = doubles.AsQueryable().Min();

Console.WriteLine("The smallest number is {0}.", min);

/*
    This code produces the following output:

    The smallest number is -2E+103.
*/
```

Remarks

The Min<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Min<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Min<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the minimum value in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Min<TSource>(IQueryable<TSource>, IComparer<TSource>)

Source: Queryable.cs

Returns the minimum value in a generic IQueryable<T>.

```
C#

public static TSource? Min<TSource> (this System.Linq.IQueryable<TSource>
source, System.Collections.Generic.IComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the minimum value of.

`comparer` IComparer<TSource>

The IComparer<T> to compare values.

Returns

TSource

The minimum value in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No object in `source` implements the IComparable or IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.MinBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.
MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)	Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.

MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.

C#

```
public static TSource? MinBy<TSource,TKey> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

TSource

The type of the elements of `source`.

TKey

The type of key to compare elements by.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the minimum value of.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

TSource

The value with the minimum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

MinBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TSource>)

Source: Queryable.cs

Returns the minimum value in a generic IQueryable<T> according to a specified key selector function.

```C#
public static TSource? MinBy<TSource,TKey> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of key to compare elements by.

Parameters

`source` IQueryable<TSource>

A sequence of values to determine the minimum value of.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IComparer<TSource>

The IComparer<T> to compare keys.

Returns

TSource

The value with the minimum key in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

ArgumentException

No key extracted from `source` implements the IComparable or IComparable<T> interface.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.OfType<TResult> (IQueryable) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Filters the elements of an IQueryable based on a specified type.

C#

```
public static System.Linq.IQueryable<TResult> OfType<TResult> (this
System.Linq.IQueryable source);
```

Type Parameters

`TResult`

The type to filter the elements of the sequence on.

Parameters

`source` IQueryable

An IQueryable whose elements to filter.

Returns

IQueryable<TResult>

A collection that contains the elements from `source` that have type `TResult`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use `OfType` to filter out elements that are not of type PropertyInfo from a list of elements of type MemberInfo.

C#

```
// Create a list of MemberInfo objects.
List<System.Reflection.MemberInfo> members =
typeof(String).GetMembers().ToList();

// Return only those items that can be cast to type PropertyInfo.
IQueryable<System.Reflection.PropertyInfo> propertiesOnly =
    members.AsQueryable().OfType<System.Reflection.PropertyInfo>();

Console.WriteLine("Members of type 'PropertyInfo' are:");
foreach (System.Reflection.PropertyInfo pi in propertiesOnly)
    Console.WriteLine(pi.Name);

/*
    This code produces the following output:

    Members of type 'PropertyInfo' are:
    Chars
    Length
*/
```

Remarks

The `OfType` method generates a MethodCallExpression that represents calling `OfType` itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling `OfType` depends on the implementation of the type of the `source` parameter. The expected behavior is that it filters out any elements in `source` that are not of type `TResult`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Queryable.Order Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

Order<T>(IQueryable<T>)	Sorts the elements of a sequence in ascending order.
Order<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in ascending order.

Order<T>(IQueryable<T>)

Source: Queryable.cs

Sorts the elements of a sequence in ascending order.

C#

```
public static System.Linq.IOrderedQueryable<T> Order<T> (this
System.Linq.IQueryable<T> source);
```

Type Parameters

T

The type of the elements of `source`.

Parameters

source IQueryable<T>

A sequence of values to order.

Returns

IOrderedQueryable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Order<T>(IQueryable<T>) method generates a MethodCallExpression that represents calling Order<T>(IEnumerable<T>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling Order<T>(IEnumerable<T>) depends on the implementation of the `source` parameter. The expected behavior is that it sorts the elements of `source` by itself.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Order<T>(IQueryable<T>, IComparer<T>)

Source: Queryable.cs

Sorts the elements of a sequence in ascending order.

```
C#
```

```
public static System.Linq.IOrderedQueryable<T> Order<T> (this
System.Linq.IQueryable<T> source, System.Collections.Generic.IComparer<T>
comparer);
```

Type Parameters

`T`

The type of the elements of `source`.

Parameters

`source` IQueryable<T>

A sequence of values to order.

`comparer` IComparer<T>

An IComparer<T> to compare elements.

Returns

IOrderedQueryable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Order<T>(IQueryable<T>) method generates a MethodCallExpression that represents calling Order<T>(IEnumerable<T>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling Order<T>(IEnumerable<T>) depends on the implementation of the `source` parameter. The expected behavior is that it sorts the elements of `source` by itself.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Queryable.OrderBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Sorts the elements of a sequence in ascending order.

Overloads

Expand table

OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in ascending order by using a specified comparer.
OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in ascending order according to a key.

OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)

Source: Queryable.cs

Sorts the elements of a sequence in ascending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedQueryable<TSource>
OrderBy<TSource,TKey> (this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function that is represented by `keySelector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to order.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `comparer` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) method generates a MethodCallExpression that represents calling OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling

CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it sorts the elements of `source` based on the key obtained by invoking `keySelector` on each element of `source`. The `comparer` parameter is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Sorts the elements of a sequence in ascending order according to a key.

C#

```
public static System.Linq.IOrderedQueryable<TSource>
OrderBy<TSource,TKey> (this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function that is represented by `keySelector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to order.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from an element.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use OrderBy<TSource,TKey> (IQueryable<TSource>, Expression<Func<TSource,TKey>>) to sort the elements of a sequence.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void OrderByEx1()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };

    // Sort the Pet objects in the array by Pet.Age.
```

```
    IEnumerable<Pet> query =
        pets.AsQueryable().OrderBy(pet => pet.Age);

    foreach (Pet pet in query)
        Console.WriteLine("{0} - {1}", pet.Name, pet.Age);
}

/*
    This code produces the following output:

    Whiskers - 1
    Boots - 4
    Barley - 8
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) method generates a MethodCallExpression that represents calling OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling OrderBy<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it sorts the elements of `source` based on the key obtained by invoking `keySelector` on each element of `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.OrderByDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Sorts the elements of a sequence in descending order.

Overloads

Expand table

OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Sorts the elements of a sequence in descending order by using a specified comparer.
OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>)	Sorts the elements of a sequence in descending order according to a key.

OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)

Source: Queryable.cs

Sorts the elements of a sequence in descending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedQueryable<TSource>
OrderByDescending<TSource,TKey> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,TKey>>
keySelector, System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function that is represented by `keySelector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to order.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from an element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `comparer` is `null`.

Examples

The following code example demonstrates how to use OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) to sort the elements of a sequence in descending order by using a custom comparer.

C#

```
/// <summary>
/// This IComparer class sorts by the fractional part of the decimal
number.
```

```
/// </summary>
public class SpecialComparer : IComparer<decimal>
{
    /// <summary>
    /// Compare two decimal numbers by their fractional parts.
    /// </summary>
    /// <param name="d1">The first decimal to compare.</param>
    /// <param name="d2">The second decimal to compare.</param>
    /// <returns>1 if the first decimal's fractional part
    /// is greater than the second decimal's fractional part,
    /// -1 if the first decimal's fractional
    /// part is less than the second decimal's fractional part,
    /// or the result of calling Decimal.Compare()
    /// if the fractional parts are equal.</returns>
    public int Compare(decimal d1, decimal d2)
    {
        decimal fractional1, fractional2;

        // Get the fractional part of the first number.
        try
        {
            fractional1 = decimal.Remainder(d1, decimal.Floor(d1));
        }
        catch (DivideByZeroException)
        {
            fractional1 = d1;
        }
        // Get the fractional part of the second number.
        try
        {
            fractional2 = decimal.Remainder(d2, decimal.Floor(d2));
        }
        catch (DivideByZeroException)
        {
            fractional2 = d2;
        }

        if (fractional1 == fractional2)
            return Decimal.Compare(d1, d2);
        else if (fractional1 > fractional2)
            return 1;
        else
            return -1;
    }
}

public static void OrderByDescendingEx1()
{
    List<decimal> decimals =
        new List<decimal> { 6.2m, 8.3m, 0.5m, 1.3m, 6.3m, 9.7m };

    // Sort the decimal values in descending order
    // by using a custom comparer.
    IEnumerable<decimal> query =
        decimals.AsQueryable()
```

```
        .OrderByDescending(num => num, new SpecialComparer());

    foreach (decimal num in query)
        Console.WriteLine(num);
}

/*
    This code produces the following output:

    9.7
    0.5
    8.3
    6.3
    1.3
    6.2
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) method generates a MethodCallExpression that represents calling OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it sorts the elements of `source` in descending order, based on the key obtained by invoking `keySelector` on each element of `source`. The `comparer` parameter is used to compare keys.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

OrderByDescending<TSource,TKey> (IQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Sorts the elements of a sequence in descending order according to a key.

C#

```
public static System.Linq.IOrderedQueryable<TSource>
OrderByDescending<TSource,TKey> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,TKey>>
keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function that is represented by `keySelector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to order.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from an element.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) method generates a MethodCallExpression that represents calling OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling OrderByDescending<TSource,TKey>(IQueryable<TSource>, Expression<Func<TSource,TKey>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it sorts the elements of `source` in descending order, based on the key obtained by invoking `keySelector` on each element of `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.OrderDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

OrderDescending<T>(IQueryable<T>)	Sorts the elements of a sequence in descending order.
OrderDescending<T>(IQueryable<T>, IComparer<T>)	Sorts the elements of a sequence in descending order.

OrderDescending<T>(IQueryable<T>)

Source: Queryable.cs

Sorts the elements of a sequence in descending order.

C#

```
public static System.Linq.IOrderedQueryable<T> OrderDescending<T> (this
System.Linq.IQueryable<T> source);
```

Type Parameters

T

The type of the elements of `source`.

Parameters

source IQueryable<T>

A sequence of values to order.

Returns

IOrderedQueryable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Order<T>(IQueryable<T>) method generates a MethodCallExpression that represents calling Order<T>(IEnumerable<T>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling Order<T>(IEnumerable<T>) depends on the implementation of the `source` parameter. The expected behavior is that it sorts the elements of `source` by itself.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

OrderDescending<T>(IQueryable<T>, IComparer<T>)

Source: Queryable.cs

Sorts the elements of a sequence in descending order.

```
C#

public static System.Linq.IOrderedQueryable<T> OrderDescending<T> (this
System.Linq.IQueryable<T> source, System.Collections.Generic.IComparer<T>
comparer);
```

Type Parameters

T

The type of the elements of `source`.

Parameters

source IQueryable<T>

A sequence of values to order.

comparer IComparer<T>

An IComparer<T> to compare elements.

Returns

IOrderedQueryable<T>

An IOrderedEnumerable<TElement> whose elements are sorted.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Order<T>(IQueryable<T>) method generates a MethodCallExpression that represents calling Order<T>(IEnumerable<T>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of

the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling Order<T>(IEnumerable<T>) depends on the implementation of the `source` parameter. The expected behavior is that it sorts the elements of `source` by itself.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	7, 8, 9

Queryable.Prepend<TSource> (IQueryable<TSource>, TSource) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Returns a new queryable sequence that contains the elements from `source` plus the specified `element` prepended at the beginning.

C#

```
public static System.Linq.IQueryable<TSource> Prepend<TSource> (this
System.Linq.IQueryable<TSource> source, TSource element);
```

Type Parameters

`TSource`

The type of the elements in the queryable sequence.

Parameters

`source` IQueryable<TSource>

A queryable sequence.

`element` TSource

An element of type `TSource` to prepend to `source`.

Returns

IQueryable<TSource>

A new queryable sequence that contains the elements from `source` plus the specified `element` prepended at the beginning.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Queryable.Reverse<TSource> (IQueryable<TSource>) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Inverts the order of the elements in a sequence.

C#

```
public static System.Linq.IQueryable<TSource> Reverse<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values to reverse.

Returns

IQueryable<TSource>

An IQueryable<T> whose elements correspond to those of the input sequence in reverse order.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Reverse<TSource>(IQueryable<TSource>) to reverse the order of elements in an array.

```C#
char[] apple = { 'a', 'p', 'p', 'l', 'e' };

// Reverse the order of the characters in the collection.
IQueryable<char> reversed = apple.AsQueryable().Reverse();

foreach (char chr in reversed)
    Console.Write(chr + " ");
Console.WriteLine();

/*
    This code produces the following output:

    e l p p a
*/
```

Remarks

The Reverse<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Reverse<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Reverse<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it reverses the order of the elements in `source`.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Queryable.Select Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Projects each element of a sequence into a new form.

Overloads

Expand table

Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>)	Projects each element of a sequence into a new form by incorporating the element's index.
Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>)	Projects each element of a sequence into a new form.

Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>)

Source: Queryable.cs

Projects each element of a sequence into a new form by incorporating the element's index.

C#

```
public static System.Linq.IQueryable<TResult> Select<TSource,TResult>
(this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int,TResult>> selector);
```

Type Parameters

TSource

The type of the elements of `source`.

`TResult`

The type of the value returned by the function represented by `selector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to project.

`selector` Expression<Func<TSource,Int32,TResult>>

A projection function to apply to each element.

Returns

IQueryable<TResult>

An IQueryable<T> whose elements are the result of invoking a projection function on each element of `source`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>) to project over a sequence of values and use the index of each element in the projected form.

C#

```
string[] fruits = { "apple", "banana", "mango", "orange",
                    "passionfruit", "grape" };

// Project an anonymous type that contains the
// index of the string in the source array, and
// a string that contains the same number of characters
// as the string's index in the source array.
var query =
    fruits.AsQueryable()
    .Select((fruit, index) =>
                new { index, str = fruit.Substring(0, index) });
```

```
foreach (var obj in query)
    Console.WriteLine("{0}", obj);

/*
    This code produces the following output:

    { index = 0, str =  }
    { index = 1, str = b }
    { index = 2, str = ma }
    { index = 3, str = ora }
    { index = 4, str = pass }
    { index = 5, str = grape }
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>) method generates a MethodCallExpression that represents calling Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,TResult>>) depend on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` to project it into a different form.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

Select<TSource,TResult> (IQueryable<TSource>, Expression<Func<TSource,TResult>>)

Source: Queryable.cs

Projects each element of a sequence into a new form.

C#

```
public static System.Linq.IQueryable<TResult> Select<TSource,TResult>
(this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TResult>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the value returned by the function represented by `selector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to project.

`selector` Expression<Func<TSource,TResult>>

A projection function to apply to each element.

Returns

IQueryable<TResult>

An IQueryable<T> whose elements are the result of invoking a projection function on each element of `source`.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) to project over a sequence of values.

C#

```
List<int> range =
    new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Project the square of each int value.
IEnumerable<int> squares =
    range.AsQueryable().Select(x => x * x);

foreach (int num in squares)
    Console.WriteLine(num);

/*
    This code produces the following output:

    1
    4
    9
    16
    25
    36
    49
    64
    81
    100
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) method generates a MethodCallExpression that represents calling Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Select<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` to project it into a different form.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.SelectMany Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence of type IQueryable<T>.

Overloads

Expand table

SelectMany<TSource,TCollection,TResult> (IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> and invokes a result selector function on each element therein. The resulting values from each intermediate sequence are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TCollection,TResult> (IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)	Projects each element of a sequence to an IEnumerable<T> that incorporates the index of the source element that produced it. A result selector function is invoked on each element of each intermediate sequence, and the resulting values are combined into a single, one-dimensional sequence and returned.
SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence.

SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>)	Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)

Source: Queryable.cs

Projects each element of a sequence to an IEnumerable<T> and invokes a result selector function on each element therein. The resulting values from each intermediate sequence are combined into a single, one-dimensional sequence and returned.

C#

```
public static System.Linq.IQueryable<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,System.Collections.Generi
c.IEnumerable<TCollection>>> collectionSelector,
System.Linq.Expressions.Expression<Func<TSource,TCollection,TResult>>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TCollection`

The type of the intermediate elements collected by the function represented by `collectionSelector`.

`TResult`

The type of the elements of the resulting sequence.

Parameters

`source` IQueryable<TSource>

A sequence of values to project.

`collectionSelector` Expression<Func<TSource,IEnumerable<TCollection>>>

A projection function to apply to each element of the input sequence.

`resultSelector` Expression<Func<TSource,TCollection,TResult>>

A projection function to apply to each element of each intermediate sequence.

Returns

IQueryable<TResult>

An IQueryable<T> whose elements are the result of invoking the one-to-many projection function `collectionSelector` on each element of `source` and then mapping each of those sequence elements and their corresponding `source` element to a result element.

Exceptions

ArgumentNullException

`source` or `collectionSelector` or `resultSelector` is `null`.

Examples

The following code example demonstrates how to use SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) to perform a one-to-many projection over an array. This example uses a result selector function to keep the source element that corresponds to each intermediate sequence in scope for the final call to `Select`.

C#

```
class PetOwner
{
    public string Name { get; set; }
    public List<Pet> Pets { get; set; }
}

class Pet
{
    public string Name { get; set; }
    public string Breed { get; set; }
}

public static void SelectManyEx3()
{
    PetOwner[] petOwners =
        { new PetOwner { Name="Higa",
              Pets = new List<Pet>{
                  new Pet { Name="Scruffy", Breed="Poodle" },
                  new Pet { Name="Sam", Breed="Hound" } } },
          new PetOwner { Name="Ashkenazi",
              Pets = new List<Pet>{
                  new Pet { Name="Walker", Breed="Collie" },
                  new Pet { Name="Sugar", Breed="Poodle" } } },
          new PetOwner { Name="Price",
              Pets = new List<Pet>{
                  new Pet { Name="Scratches", Breed="Dachshund" },
                  new Pet { Name="Diesel", Breed="Collie" } } },
          new PetOwner { Name="Hines",
              Pets = new List<Pet>{
                  new Pet { Name="Dusty", Breed="Collie" } } }
        };

    // This query demonstrates how to obtain a sequence of
    // the names of all the pets whose breed is "Collie", while
    // keeping an association with the owner that owns the pet.
    var query =
        petOwners.AsQueryable()
        // Create a sequence of ALL the Pet objects. Then
        // project an anonymous type that consists of each
        // Pet in the new sequence and the PetOwner object
        // from the initial array that corresponds to that pet.
       .SelectMany(owner => owner.Pets,
                   (owner, pet) => new { owner, pet })
        // Filter the sequence of anonymous types to only
        // keep pets whose breed is "Collie".
        .Where(ownerAndPet => ownerAndPet.pet.Breed == "Collie")
        // Project an anonymous type that consists
        // of the pet owner's name and the pet's name.
        .Select(ownerAndPet => new
        {
            Owner = ownerAndPet.owner.Name,
            Pet = ownerAndPet.pet.Name
```

```
        });

    // Print the results.
    foreach (var obj in query)
        Console.WriteLine(obj);
}

/* This code produces the following output:

    { Owner = Ashkenazi, Pet = Walker }
    { Owner = Price, Pet = Diesel }
    { Owner = Hines, Pet = Dusty }
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) method generates a MethodCallExpression that represents calling SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `collectionSelector` on each element of `source` to project it into an enumerable form. Then the function represented by `resultSelector` is invoked on each element in each intermediate sequence. The resulting values are concatenated into a single, one-dimensional sequence.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>)

Source: Queryable.cs

Projects each element of a sequence to an IEnumerable<T> that incorporates the index of the source element that produced it. A result selector function is invoked on each element of each intermediate sequence, and the resulting values are combined into a single, one-dimensional sequence and returned.

C#

```
public static System.Linq.IQueryable<TResult>
SelectMany<TSource,TCollection,TResult> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int,System.Collections.Ge
neric.IEnumerable<TCollection>>> collectionSelector,
System.Linq.Expressions.Expression<Func<TSource,TCollection,TResult>>
resultSelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TCollection`

The type of the intermediate elements collected by the function represented by `collectionSelector`.

`TResult`

The type of the elements of the resulting sequence.

Parameters

`source` IQueryable<TSource>

A sequence of values to project.

`collectionSelector` Expression<Func<TSource,Int32,IEnumerable<TCollection>>>

A projection function to apply to each element of the input sequence; the second parameter of this function represents the index of the source element.

`resultSelector` Expression<Func<TSource,TCollection,TResult>>

A projection function to apply to each element of each intermediate sequence.

Returns

IQueryable<TResult>

An IQueryable<T> whose elements are the result of invoking the one-to-many projection function `collectionSelector` on each element of `source` and then mapping each of those sequence elements and their corresponding `source` element to a result element.

Exceptions

ArgumentNullException

`source` or `collectionSelector` or `resultSelector` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) method generates a MethodCallExpression that represents calling SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SelectMany<TSource,TCollection,TResult>(IQueryable<TSource>, Expression<Func<TSource, Int32,IEnumerable<TCollection>>>, Expression<Func<TSource,TCollection, TResult>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `collectionSelector` on each element of `source` to project it into an enumerable form. Each enumerable result incorporates the source element's index. Then the function represented by `resultSelector` is invoked on each element in each intermediate sequence. The resulting values are concatenated into a single, one-dimensional sequence.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SelectMany<TSource,TResult>(IQueryable<TSource>,

Expression<Func<TSource,IEnumerable<TResult>>>)

Source: Queryable.cs

Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence.

```
C#

public static System.Linq.IQueryable<TResult> SelectMany<TSource,TResult>
(this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,System.Collections.Generi
c.IEnumerable<TResult>>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the elements of the sequence returned by the function represented by `selector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to project.

`selector` Expression<Func<TSource,IEnumerable<TResult>>>

A projection function to apply to each element.

Returns

IQueryable<TResult>

An IQueryable<T> whose elements are the result of invoking a one-to-many projection function on each element of the input sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>) to perform a one-to-many projection over an array.

C#

```
class PetOwner
{
    public string Name { get; set; }
    public List<String> Pets { get; set; }
}

public static void SelectManyEx1()
{
    PetOwner[] petOwners =
        { new PetOwner { Name="Higa, Sidney",
              Pets = new List<string>{ "Scruffy", "Sam" } },
          new PetOwner { Name="Ashkenazi, Ronen",
              Pets = new List<string>{ "Walker", "Sugar" } },
          new PetOwner { Name="Price, Vernette",
              Pets = new List<string>{ "Scratches", "Diesel" } } };

    // Query using SelectMany().
    IEnumerable<string> query1 =
        petOwners.AsQueryable().SelectMany(petOwner => petOwner.Pets);

    Console.WriteLine("Using SelectMany():");

    // Only one foreach loop is required to iterate through the
    // results because it is a one-dimensional collection.
    foreach (string pet in query1)
        Console.WriteLine(pet);

    // This code shows how to use Select() instead of SelectMany().
    IEnumerable<List<String>> query2 =
        petOwners.AsQueryable().Select(petOwner => petOwner.Pets);

    Console.WriteLine("\nUsing Select():");

    // Notice that two foreach loops are required to iterate through
    // the results because the query returns a collection of arrays.
    foreach (List<String> petList in query2)
    {
        foreach (string pet in petList)
        {
            Console.WriteLine(pet);
        }
        Console.WriteLine();
```

```
    }
}

/*
    This code produces the following output:

    Using SelectMany():
    Scruffy
    Sam
    Walker
    Sugar
    Scratches
    Diesel

    Using Select():
    Scruffy
    Sam

    Walker
    Sugar

    Scratches
    Diesel
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>) method generates a MethodCallExpression that represents calling SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,IEnumerable<TResult>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` to project it into an enumerable form. It then concatenates the enumerable results into a single, one-dimensional sequence.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SelectMany<TSource,TResult> (IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>)

Source: Queryable.cs

Projects each element of a sequence to an IEnumerable<T> and combines the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.

C#

```
public static System.Linq.IQueryable<TResult> SelectMany<TSource,TResult>
(this System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int,System.Collections.Ge
neric.IEnumerable<TResult>>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TResult`

The type of the elements of the sequence returned by the function represented by `selector`.

Parameters

`source` IQueryable<TSource>

A sequence of values to project.

`selector` Expression<Func<TSource,Int32,IEnumerable<TResult>>>

A projection function to apply to each element; the second parameter of this function represents the index of the source element.

Returns

IQueryable<TResult>

An IQueryable<T> whose elements are the result of invoking a one-to-many projection function on each element of the input sequence.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>) to perform a one-to-many projection over an array and use the index of each source element.

C#

```
class PetOwner
{
    public string Name { get; set; }
    public List<string> Pets { get; set; }
}

public static void SelectManyEx2()
{
    PetOwner[] petOwners =
        { new PetOwner { Name="Higa, Sidney",
              Pets = new List<string>{ "Scruffy", "Sam" } },
          new PetOwner { Name="Ashkenazi, Ronen",
              Pets = new List<string>{ "Walker", "Sugar" } },
          new PetOwner { Name="Price, Vernette",
              Pets = new List<string>{ "Scratches", "Diesel" } },
          new PetOwner { Name="Hines, Patrick",
              Pets = new List<string>{ "Dusty" } } };
```

```
    // For each PetOwner element in the source array,
    // project a sequence of strings where each string
    // consists of the index of the PetOwner element in the
    // source array and the name of each pet in PetOwner.Pets.
    IEnumerable<string> query =
        petOwners.AsQueryable()
        .SelectMany(
        (petOwner, index) => petOwner.Pets.Select(pet => index + pet)
        );

    foreach (string pet in query)
        Console.WriteLine(pet);
}

// This code produces the following output:
//
// 0Scruffy
// 0Sam
// 1Walker
// 1Sugar
// 2Scratches
// 2Diesel
// 3Dusty
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>) method generates a MethodCallExpression that represents calling SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SelectMany<TSource,TResult>(IQueryable<TSource>, Expression<Func<TSource,Int32,IEnumerable<TResult>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` to project it into an enumerable form. Each enumerable result incorporates the index of the source element. It then concatenates the enumerable results into a single, one-dimensional sequence.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.SequenceEqual Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Determines whether two sequences are equal.

Overloads

Expand table

SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Determines whether two sequences are equal by using the default equality comparer to compare elements.
SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Determines whether two sequences are equal by using a specified IEqualityComparer<T> to compare elements.

SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>)

Source: Queryable.cs

Determines whether two sequences are equal by using the default equality comparer to compare elements.

C#

```
public static bool SequenceEqual<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose elements to compare to those of `source2`.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose elements to compare to those of the first sequence.

Returns

Boolean

`true` if the two source sequences are of equal length and their corresponding elements compare equal; otherwise, `false`.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Examples

The following code example demonstrates how to use SequenceEqual<TSource> (IQueryable<TSource>, IEnumerable<TSource>) to determine whether two sequences are equal. In this example the sequences are equal.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void SequenceEqualEx1()
{
    Pet pet1 = new Pet { Name = "Turbo", Age = 2 };
    Pet pet2 = new Pet { Name = "Peanut", Age = 8 };

    // Create two lists of pets.
    List<Pet> pets1 = new List<Pet> { pet1, pet2 };
    List<Pet> pets2 = new List<Pet> { pet1, pet2 };
```

```
    // Determine if the lists are equal.
    bool equal = pets1.AsQueryable().SequenceEqual(pets2);

    Console.WriteLine(
        "The lists {0} equal.",
        equal ? "are" : "are not");
}

/*
    This code produces the following output:

    The lists are equal.
*/
```

The following code example compares two sequences that are not equal.

C#

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void SequenceEqualEx2()
{
    Pet pet1 = new Pet() { Name = "Turbo", Age = 2 };
    Pet pet2 = new Pet() { Name = "Peanut", Age = 8 };

    // Create two lists of pets.
    List<Pet> pets1 = new List<Pet> { pet1, pet2 };
    List<Pet> pets2 = new List<Pet> {
        new Pet { Name = "Turbo", Age = 2 },
        new Pet { Name = "Peanut", Age = 8 }
    };

    // Determine if the lists are equal.
    bool equal = pets1.AsQueryable().SequenceEqual(pets2);

    Console.WriteLine("The lists {0} equal.", equal ? "are" : "are NOT");
}

/*
    This code produces the following output:

    The lists are NOT equal.
*/
```

Remarks

The SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>) method generates a MethodCallExpression that represents calling SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that it determines if the two source sequences are equal.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Queryable.cs

Determines whether two sequences are equal by using a specified IEqualityComparer<T> to compare elements.

```
C#
```

```
public static bool SequenceEqual<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose elements to compare to those of `source2`.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose elements to compare to those of the first sequence.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to use to compare elements.

Returns

Boolean

`true` if the two source sequences are of equal length and their corresponding elements compare equal; otherwise, `false`.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Remarks

The SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method generates a MethodCallExpression that represents calling SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>

(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SequenceEqual<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that it determines if the two source sequences are equal by using `comparer` to compare elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Single Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns a single, specific element of a sequence.

Overloads

Expand table

Single<TSource>(IQueryable<TSource>)	Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.
Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

Single<TSource>(IQueryable<TSource>)

Source: Queryable.cs

Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.

C#

```
public static TSource Single<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return the single element of.

Returns

TSource

The single element of the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` has more than one element.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use Single<TSource> (IQueryable<TSource>) to select the only element of an array.

C#

```
// Create two arrays.
string[] fruits1 = { "orange" };
string[] fruits2 = { "orange", "apple" };

// Get the only item in the first array.
string fruit1 = fruits1.AsQueryable().Single();

Console.WriteLine("First query: " + fruit1);

try
{
    // Try to get the only item in the second array.
    string fruit2 = fruits2.AsQueryable().Single();
    Console.WriteLine("Second query: " + fruit2);
}
catch (System.InvalidOperationException)
{
    Console.WriteLine(
```

```
        "Second query: The collection does not contain exactly one
element."
        );
}

/*
    This code produces the following output:

    First query: orange
    Second query: The collection does not contain exactly one element
*/
```

Remarks

The Single<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling Single<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Single<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the only element in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

```
C#

public static TSource Single<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return a single element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test an element for a condition.

Returns

TSource

The single element of the input sequence that satisfies the condition in `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

No element satisfies the condition in `predicate`.

-or-

More than one element satisfies the condition in `predicate`.

-or-

The source sequence is empty.

Examples

The following code example demonstrates how to use Single<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to select the only element of an array that satisfies a condition.

C#

```
string[] fruits = { "apple", "banana", "mango",
                      "orange", "passionfruit", "grape" };

// Get the only string in the array whose length is greater than 10.
string fruit1 = fruits.AsQueryable().Single(fruit => fruit.Length > 10);

Console.WriteLine("First Query: " + fruit1);

try
{
    // Try to get the only string in the array
    // whose length is greater than 15.
    string fruit2 = fruits.AsQueryable().Single(fruit => fruit.Length >
15);
    Console.WriteLine("Second Query: " + fruit2);
}
catch (System.InvalidOperationException)
{
    Console.Write("Second Query: The collection does not contain ");
    Console.WriteLine("exactly one element whose length is greater than
15.");
}

/*
    This code produces the following output:

    First Query: passionfruit
    Second Query: The collection does not contain exactly one
    element whose length is greater than 15.
 */
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Single<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the only element in `source` that satisfies the condition specified by `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.SingleOrDefault Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns a single, specific element of a sequence, or a default value if no such element is found.

Overloads

Expand table

SingleOrDefault<TSource> (IQueryable<TSource>, TSource)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.
SingleOrDefault<TSource> (IQueryable<TSource>)	Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.
SingleOrDefault<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

SingleOrDefault<TSource> (IQueryable<TSource>, TSource)

Source: Queryable.cs

Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

```C#
public static TSource SingleOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source, TSource defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IEnumerable<T> to return the single element of.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

The single element of the input sequence, or `defaultValue` if the sequence contains no elements.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

The input sequence contains more than one element.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

SingleOrDefault<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>, TSource)

Source: Queryable.cs

Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

```
C#

public static TSource SingleOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate, TSource
defaultValue);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IEnumerable<T> to return a single element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test an element for a condition.

`defaultValue` TSource

The default value to return if the sequence is empty.

Returns

TSource

The single element of the input sequence that satisfies the condition, or `defaultValue` if no such element is found.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

More than one element satisfies the condition in `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

SingleOrDefault<TSource> (IQueryable<TSource>)

Source: Queryable.cs

Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

C#

```
public static TSource? SingleOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return the single element of.

Returns

TSource

The single element of the input sequence, or `default(TSource)` if the sequence contains no elements.

Exceptions

ArgumentNullException

`source` is `null`.

InvalidOperationException

`source` has more than one element.

Examples

The following code example demonstrates how to use SingleOrDefault<TSource>(IQueryable<TSource>) to select the only element of an array. The second query demonstrates that SingleOrDefault<TSource>(IQueryable<TSource>) returns a default value when the sequence does not contain exactly one element.

C#

```
// Create two arrays. The second is empty.
string[] fruits1 = { "orange" };
string[] fruits2 = { };

// Get the only item in the first array, or else
// the default value for type string (null).
string fruit1 = fruits1.AsQueryable().SingleOrDefault();
Console.WriteLine("First Query: " + fruit1);

// Get the only item in the second array, or else
// the default value for type string (null).
string fruit2 = fruits2.AsQueryable().SingleOrDefault();
Console.WriteLine("Second Query: " +
    (String.IsNullOrEmpty(fruit2) ? "No such string!" : fruit2));

/*
    This code produces the following output:

    First Query: orange
```

```
    Second Query: No such string!
*/
```

Sometimes the value of `default(TSource)` is not the default value that you want to use if the collection contains no elements. Instead of checking the result for the unwanted default value and then changing it if necessary, you can use the DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method to specify the default value that you want to use if the collection is empty. Then, call Single<TSource>(IQueryable<TSource>) to obtain the element. The following code example uses both techniques to obtain a default value of 1 if a collection of page numbers is empty. Because the default value for an integer is 0, which is not usually a valid page number, the default value must be specified as 1 instead. The first result variable is checked for the unwanted default value after the query is completed. The second result variable is obtained by calling DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) to specify a default value of 1.

C#

```
int[] pageNumbers = { };

// Setting the default value to 1 after the query.
int pageNumber1 = pageNumbers.AsQueryable().SingleOrDefault();
if (pageNumber1 == 0)
{
    pageNumber1 = 1;
}
Console.WriteLine("The value of the pageNumber1 variable is {0}",
pageNumber1);

// Setting the default value to 1 by using DefaultIfEmpty() in the query.
int pageNumber2 = pageNumbers.AsQueryable().DefaultIfEmpty(1).Single();
Console.WriteLine("The value of the pageNumber2 variable is {0}",
pageNumber2);

/*
 This code produces the following output:

 The value of the pageNumber1 variable is 1
 The value of the pageNumber2 variable is 1
*/
```

Remarks

The SingleOrDefault<TSource>(IQueryable<TSource>) method generates a MethodCallExpression that represents calling SingleOrDefault<TSource>(IQueryable<TSource>) itself as a constructed generic method. It then passes the

MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SingleOrDefault<TSource>(IQueryable<TSource>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the only element in `source`, or a default value if `source` is empty.

The SingleOrDefault method does not provide a way to specify a default value. If you want to specify a default value other than `default(TSource)`, use the DefaultIfEmpty<TSource>(IQueryable<TSource>, TSource) method as described in the Example section.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

C#

```
public static TSource? SingleOrDefault<TSource> (this
System.Linq.IQueryable<TSource> source,
```

```
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return a single element from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test an element for a condition.

Returns

TSource

The single element of the input sequence that satisfies the condition in `predicate`, or `default`(`TSource`) if no such element is found.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

InvalidOperationException

More than one element satisfies the condition in `predicate`.

Examples

The following code example demonstrates how to use SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to select the only element of an array that satisfies a condition. The second query demonstrates that SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) returns a default value when the sequence does not contain exactly one element that satisfies the condition.

```
C#
```

```
string[] fruits = { "apple", "banana", "mango",
                      "orange", "passionfruit", "grape" };

// Get the single string in the array whose length is greater
// than 10, or else the default value for type string (null).
string fruit1 =
    fruits.AsQueryable().SingleOrDefault(fruit => fruit.Length > 10);
Console.WriteLine("First Query: " + fruit1);

// Get the single string in the array whose length is greater
// than 15, or else the default value for type string (null).
string fruit2 =
   fruits.AsQueryable().SingleOrDefault(fruit => fruit.Length > 15);
Console.WriteLine("Second Query: " +
    (String.IsNullOrEmpty(fruit2) ? "No such string!" : fruit2));

/*
    This code produces the following output:

    First Query: passionfruit
    Second Query: No such string!
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SingleOrDefault<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the only element in `source` that satisfies the condition specified by `predicate`, or a default value if no such element exists.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Skip<TSource> (IQueryable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Bypasses a specified number of elements in a sequence and then returns the remaining elements.

C#

```
public static System.Linq.IQueryable<TSource> Skip<TSource> (this
System.Linq.IQueryable<TSource> source, int count);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return elements from.

`count` Int32

The number of elements to skip before returning the remaining elements.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements that occur after the specified index in the input sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Skip<TSource>(IQueryable<TSource>, Int32) to skip a specified number of elements in a sorted array and return the remaining elements.

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

// Sort the grades in descending order and
// get all except the first three.
IEnumerable<int> lowerGrades =
    grades.AsQueryable().OrderByDescending(g => g).Skip(3);

Console.WriteLine("All grades except the top three are:");
foreach (int grade in lowerGrades)
    Console.WriteLine(grade);

/*
    This code produces the following output:

    All grades except the top three are:
    82
    70
    59
    56
*/
```

Remarks

The Skip<TSource>(IQueryable<TSource>, Int32) method generates a MethodCallExpression that represents calling Skip<TSource>(IQueryable<TSource>, Int32) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Skip<TSource>(IQueryable<TSource>, Int32) depends on the implementation of the type of the `source` parameter. The expected behavior is that it skips the first `count` elements in `source` and returns the remaining elements.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.SkipLast<TSource> (IQueryable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Returns a new queryable sequence that contains the elements from `source` with the last `count` elements of the source queryable sequence omitted.

C#

```
public static System.Linq.IQueryable<TSource> SkipLast<TSource> (this
System.Linq.IQueryable<TSource> source, int count);
```

Type Parameters

`TSource`

The type of the elements in the queryable sequence.

Parameters

`source` IQueryable<TSource>

A queryable sequence.

`count` Int32

The number of elements to omit from the end of the queryable sequence.

Returns

IQueryable<TSource>

A new queryable sequence that contains the elements from `source` minus `count` elements from the end of the queryable sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

If `count` is not a positive number, this method returns an identical copy of the `source` queryable sequence.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Queryable.SkipWhile Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.

Overloads

Expand table

SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.
SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.

C#

```
public static System.Linq.IQueryable<TSource> SkipWhile<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return elements from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements from `source` starting at the first element in the linear series that does not pass the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use SkipWhile<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to skip elements of an array as long as a condition is true.

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

// Get all grades less than 80 by first
// sorting the grades in descending order and then
// taking all the grades after the first grade
// that is less than 80.
IEnumerable<int> lowerGrades =
    grades.AsQueryable()
    .OrderByDescending(grade => grade)
    .SkipWhile(grade => grade >= 80);

Console.WriteLine("All grades below 80:");
foreach (int grade in lowerGrades)
```

```
    Console.WriteLine(grade);

/*
    This code produces the following output:

    All grades below 80:
    70
    59
    56
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it applies `predicate` to each element in `source` until it finds an element for which `predicate` returns false. That element and all the remaining elements are returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)

Source: Queryable.cs

Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.

C#

```
public static System.Linq.IQueryable<TSource> SkipWhile<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to return elements from.

`predicate` Expression<Func<TSource,Int32,Boolean>>

A function to test each element for a condition; the second parameter of this function represents the index of the source element.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements from `source` starting at the first element in the linear series that does not pass the test specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use SkipWhile<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) to skip elements of an array as long as a condition that depends on the element's index is true.

C#

```
int[] amounts = { 5000, 2500, 9000, 8000,
                    6500, 4000, 1500, 5500 };

// Skip over amounts in the array until the first amount
// that is less than or equal to the product of its
// index in the array and 1000. Take the remaining items.
IEnumerable<int> query =
    amounts.AsQueryable()
    .SkipWhile((amount, index) => amount > index * 1000);

foreach (int amount in query)
    Console.WriteLine(amount);

/*
    This code produces the following output:

    4000
    1500
    5500
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) method generates a MethodCallExpression that represents calling SkipWhile<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the

CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling SkipWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it applies `predicate` to each element in `source` until it finds an element for which `predicate` returns false. That element and all the remaining elements are returned. The index of each source element is provided as the second argument to `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Sum Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Computes the sum of a sequence of numeric values.

Overloads

Expand table

Sum(IQueryable<Decimal>)	Computes the sum of a sequence of Decimal values.
Sum(IQueryable<Double>)	Computes the sum of a sequence of Double values.
Sum(IQueryable<Int32>)	Computes the sum of a sequence of Int32 values.
Sum(IQueryable<Int64>)	Computes the sum of a sequence of Int64 values.
Sum(IQueryable<Nullable<Decimal>>)	Computes the sum of a sequence of nullable Decimal values.
Sum(IQueryable<Nullable<Double>>)	Computes the sum of a sequence of nullable Double values.
Sum(IQueryable<Nullable<Int32>>)	Computes the sum of a sequence of nullable Int32 values.
Sum(IQueryable<Nullable<Int64>>)	Computes the sum of a sequence of nullable Int64 values.
Sum(IQueryable<Nullable<Single>>)	Computes the sum of a sequence of nullable Single values.
Sum(IQueryable<Single>)	Computes the sum of a sequence of Single values.

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)	Computes the sum of the sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)	Computes the sum of the sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)	Computes the sum of the sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)	Computes the sum of the sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)	Computes the sum of the sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)	Computes the sum of the sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)	Computes the sum of the sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)	Computes the sum of the sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.
Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)	Computes the sum of the sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)	Computes the sum of the sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.

Sum(IQueryable<Decimal>)

Source: Queryable.cs

Computes the sum of a sequence of Decimal values.

```
C#
public static decimal Sum (this System.Linq.IQueryable<decimal> source);
```

Parameters

`source` IQueryable<Decimal>

A sequence of Decimal values to calculate the sum of.

Returns

Decimal

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Remarks

The Sum(IQueryable<Decimal>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Decimal>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Decimal>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Double>)

Source: Queryable.cs

Computes the sum of a sequence of Double values.

C#

```
public static double Sum (this System.Linq.IQueryable<double> source);
```

Parameters

`source` IQueryable<Double>

A sequence of Double values to calculate the sum of.

Returns

Double

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Sum(IQueryable<Double>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Double>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Double>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Int32>)

Source: Queryable.cs

Computes the sum of a sequence of Int32 values.

C#

```
public static int Sum (this System.Linq.IQueryable<int> source);
```

Parameters

`source` IQueryable<Int32>

A sequence of Int32 values to calculate the sum of.

Returns

Int32

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Remarks

The Sum(IQueryable<Int32>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Int32>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Int32>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1

Product	Versions
UWP	10.0

Sum(IQueryable<Int64>)

Source: Queryable.cs

Computes the sum of a sequence of Int64 values.

```
C#
public static long Sum (this System.Linq.IQueryable<long> source);
```

Parameters

`source` IQueryable<Int64>

A sequence of Int64 values to calculate the sum of.

Returns

Int64

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Remarks

The Sum(IQueryable<Int64>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Int64>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Int64>) depends on the implementation of the

type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Nullable<Decimal>>)

Source: Queryable.cs

Computes the sum of a sequence of nullable Decimal values.

```
C#

public static decimal? Sum (this System.Linq.IQueryable<decimal?>
source);
```

Parameters

`source` IQueryable<Nullable<Decimal>>

A sequence of nullable Decimal values to calculate the sum of.

Returns

Nullable<Decimal>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException
The sum is larger than Decimal.MaxValue.

Remarks

The Sum(IQueryable<Nullable<Decimal>>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Nullable<Decimal>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Nullable<Decimal>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Nullable<Double>>)

Source: Queryable.cs

Computes the sum of a sequence of nullable Double values.

C#

```
public static double? Sum (this System.Linq.IQueryable<double?> source);
```

Parameters

`source` IQueryable<Nullable<Double>>

A sequence of nullable Double values to calculate the sum of.

Returns

Nullable<Double>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

The Sum(IQueryable<Nullable<Double>>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Nullable<Double>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Nullable<Double>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Nullable<Int32>>)

Source: Queryable.cs

Computes the sum of a sequence of nullable Int32 values.

```
C#
public static int? Sum (this System.Linq.IQueryable<int?> source);
```

Parameters

`source` IQueryable<Nullable<Int32>>

A sequence of nullable Int32 values to calculate the sum of.

Returns

Nullable<Int32>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Remarks

The Sum(IQueryable<Nullable<Int32>>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Nullable<Int32>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Nullable<Int32>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Nullable<Int64>>)

Source: Queryable.cs

Computes the sum of a sequence of nullable Int64 values.

C#

```
public static long? Sum (this System.Linq.IQueryable<long?> source);
```

Parameters

`source` IQueryable<Nullable<Int64>>

A sequence of nullable Int64 values to calculate the sum of.

Returns

Nullable<Int64>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Remarks

The Sum(IQueryable<Nullable<Int64>>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Nullable<Int64>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Nullable<Int64>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Nullable<Single>>)

Source: Queryable.cs

Computes the sum of a sequence of nullable Single values.

C#

```
public static float? Sum (this System.Linq.IQueryable<float?> source);
```

Parameters

`source` IQueryable<Nullable<Single>>

A sequence of nullable Single values to calculate the sum of.

Returns

Nullable<Single>

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Sum(IQueryable<Nullable<Single>>) to sum the values of a sequence.

```
C#

float?[] points = { null, 0, 92.83F, null, 100.0F, 37.46F, 81.1F };

float? sum = points.AsQueryable().Sum();

Console.WriteLine("Total points earned: {0}", sum);

/*
    This code produces the following output:

    Total points earned: 311.39
*/
```

Remarks

The Sum(IQueryable<Nullable<Single>>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Nullable<Single>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Nullable<Single>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum(IQueryable<Single>)

Source: Queryable.cs

Computes the sum of a sequence of Single values.

```
C#
public static float Sum (this System.Linq.IQueryable<float> source);
```

Parameters

`source` IQueryable<Single>

A sequence of Single values to calculate the sum of.

Returns

Single

The sum of the values in the sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Sum(IQueryable<Single>) to sum the values of a sequence.

```C#
List<float> numbers = new List<float> { 43.68F, 1.25F, 583.7F, 6.5F };

float sum = numbers.AsQueryable().Sum();

Console.WriteLine("The sum of the numbers is {0}.", sum);

/*
    This code produces the following output:

    The sum of the numbers is 635.13.
*/
```

Remarks

The Sum(IQueryable<Single>) method generates a MethodCallExpression that represents calling Sum(IQueryable<Single>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum(IQueryable<Single>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the sum of the values in `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>)

Source: Queryable.cs

Computes the sum of the sequence of Single values that is obtained by invoking a projection function on each element of the input sequence.

```
C#

public static float Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,float>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Single>>

A projection function to apply to each element.

Returns

Single

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected

values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Single>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>)

Source: Queryable.cs

Computes the sum of the sequence of nullable Single values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static float? Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,float?>>
```

```
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Nullable<Single>>>

A projection function to apply to each element.

Returns

Nullable<Single>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

```
C#
```

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Single>>>) depends on the implementation of

the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>)

Source: Queryable.cs

Computes the sum of the sequence of nullable Int64 values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static long? Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,long?>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Nullable<Int64>>>

A projection function to apply to each element.

Returns

Nullable<Int64>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
```

```
                new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int64>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>)

Source: Queryable.cs

Computes the sum of the sequence of nullable Double values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double? Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,double?>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Nullable<Double>>>

A projection function to apply to each element.

Returns

Nullable<Double>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Sum<TSource> (IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Double>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>)

Source: Queryable.cs

Computes the sum of the sequence of nullable Int32 values that is obtained by invoking a projection function on each element of the input sequence.

```C#
public static int? Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,int?>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Nullable<Int32>>>

A projection function to apply to each element.

Returns

Nullable<Int32>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

ⓘ **Note**

This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>

(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Int32>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>)

Source: Queryable.cs

Computes the sum of the sequence of Int64 values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static long Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,long>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Int64>>

A projection function to apply to each element.

Returns

Int64

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int64.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
```

```
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int64>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>)

Source: Queryable.cs

Computes the sum of the sequence of nullable Decimal values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static decimal? Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,decimal?>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Nullable<Decimal>>>

A projection function to apply to each element.

Returns

Nullable<Decimal>

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);
```

```
    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Nullable<Decimal>>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>)

Source: Queryable.cs

Computes the sum of the sequence of Int32 values that is obtained by invoking a projection function on each element of the input sequence.

```
C#

public static int Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,int>> selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Int32>>

A projection function to apply to each element.

Returns

Int32

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Int32.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>)

Source: Queryable.cs

Computes the sum of the sequence of Double values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static double Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,double>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Double>>

A projection function to apply to each element.

Returns

Double

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

Examples

The following code example demonstrates how to use Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}
```

```
public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);

    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Double>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>)

Source: Queryable.cs

Computes the sum of the sequence of Decimal values that is obtained by invoking a projection function on each element of the input sequence.

C#

```
public static decimal Sum<TSource> (this System.Linq.IQueryable<TSource>
source, System.Linq.Expressions.Expression<Func<TSource,decimal>>
selector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

A sequence of values of type `TSource`.

`selector` Expression<Func<TSource,Decimal>>

A projection function to apply to each element.

Returns

Decimal

The sum of the projected values.

Exceptions

ArgumentNullException

`source` or `selector` is `null`.

OverflowException

The sum is larger than Decimal.MaxValue.

Examples

The following code example demonstrates how to use Sum<TSource> (IQueryable<TSource>, Expression<Func<TSource,Double>>) to sum the projected values of a sequence.

> **Note**
>
> This code example uses an overload of the method that's different from the specific overload that this article describes. To extend the example to the overload that this article describes, change the body of the `selector` function.

C#

```
class Package
{
    public string Company { get; set; }
    public double Weight { get; set; }
}

public static void SumEx3()
{
    List<Package> packages =
        new List<Package>
            { new Package { Company = "Coho Vineyard", Weight = 25.2 },
              new Package { Company = "Lucerne Publishing", Weight = 18.7
},
              new Package { Company = "Wingtip Toys", Weight = 6.0 },
              new Package { Company = "Adventure Works", Weight = 33.8 }
};

    // Calculate the sum of all package weights.
    double totalWeight = packages.AsQueryable().Sum(pkg => pkg.Weight);
```

```
    Console.WriteLine("The total weight of the packages is: {0}",
totalWeight);
}

/*
    This code produces the following output:

    The total weight of the packages is: 83.7
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>) method generates a MethodCallExpression that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>) itself as a constructed generic method. It then passes the MethodCallExpression to the Execute<TResult>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Sum<TSource>(IQueryable<TSource>, Expression<Func<TSource,Decimal>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it invokes `selector` on each element of `source` and returns the sum of the resulting values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Take Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

Take<TSource> (IQueryable<TSource>, Int32)	Returns a specified number of contiguous elements from the start of a sequence.
Take<TSource> (IQueryable<TSource>, Range)	Returns a specified range of contiguous elements from a sequence.

Take<TSource>(IQueryable<TSource>, Int32)

Source: Queryable.cs

Returns a specified number of contiguous elements from the start of a sequence.

C#

```
public static System.Linq.IQueryable<TSource> Take<TSource> (this
System.Linq.IQueryable<TSource> source, int count);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

source IQueryable<TSource>

The sequence to return elements from.

count Int32

The number of elements to return.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the specified number of elements from the start of `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Examples

The following code example demonstrates how to use Take<TSource>(IQueryable<TSource>, Int32) to return elements from the start of a sequence.

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

// Sort the grades in descending order and take the first three.
IEnumerable<int> topThreeGrades =
    grades.AsQueryable().OrderByDescending(grade => grade).Take(3);

Console.WriteLine("The top three grades are:");
foreach (int grade in topThreeGrades)
    Console.WriteLine(grade);

/*
    This code produces the following output:

    The top three grades are:
    98
    92
    85
*/
```

Remarks

The Take<TSource>(IQueryable<TSource>, Int32) method generates a MethodCallExpression that represents calling Take<TSource>(IQueryable<TSource>, Int32) itself as a constructed generic method. It then passes the

MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Take<TSource>(IQueryable<TSource>, Int32) depends on the implementation of the type of the `source` parameter. The expected behavior is that it takes the first `count` elements from the start of `source`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Take<TSource>(IQueryable<TSource>, Range)

Source: Queryable.cs

Returns a specified range of contiguous elements from a sequence.

C#

```
public static System.Linq.IQueryable<TSource> Take<TSource> (this
System.Linq.IQueryable<TSource> source, Range range);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The sequence to return elements from.

`range` Range

The range of elements to return, which has start and end indexes either from the start or the end.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the specified `range` of elements from the `source` sequence.

Exceptions

ArgumentNullException

`source` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.TakeLast<TSource> (IQueryable<TSource>, Int32) Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Source: Queryable.cs

Returns a new queryable sequence that contains the last `count` elements from `source`.

C#

```
public static System.Linq.IQueryable<TSource> TakeLast<TSource> (this
System.Linq.IQueryable<TSource> source, int count);
```

Type Parameters

`TSource`

The type of the elements in the queryable sequence.

Parameters

`source` IQueryable<TSource>

A queryable sequence instance.

`count` Int32

The number of elements to take from the end of the queryable sequence.

Returns

IQueryable<TSource>

A new queryable sequence that contains the last `count` elements from `source`.

Exceptions

ArgumentNullException

`source` is `null`.

Remarks

If `count` is not a positive number, this method returns an empty queryable sequence.

Applies to

Product	Versions
.NET	Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Standard	2.1

Queryable.TakeWhile Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Returns elements from a sequence as long as a specified condition is true, and then skips the remaining elements.

Overloads

Expand table

TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.
TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Returns elements from a sequence as long as a specified condition is true.

TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)

Source: Queryable.cs

Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.

C#

```
public static System.Linq.IQueryable<TSource> TakeWhile<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int,bool>> predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The sequence to return elements from.

`predicate` Expression<Func<TSource,Int32,Boolean>>

A function to test each element for a condition; the second parameter of the function represents the index of the element in the source sequence.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements from the input sequence occurring before the element at which the test specified by `predicate` no longer passes.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use TakeWhile<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) to return elements from the start of a sequence as long as a condition that uses the index of the element is true.

C#

```
string[] fruits = { "apple", "passionfruit", "banana", "mango",
                    "orange", "blueberry", "grape", "strawberry" };

// Take strings from the array until a string whose length
// is less than its index in the array is found.
IEnumerable<string> query =
    fruits.AsQueryable()
    .TakeWhile((fruit, index) => fruit.Length >= index);

foreach (string fruit in query)
    Console.WriteLine(fruit);

/*
```

```
    This code produces the following output:

    apple
    passionfruit
    banana
    mango
    orange
    blueberry
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) method generates a MethodCallExpression that represents calling TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it applies `predicate` to each element in `source` until it finds an element for which `predicate` returns `false`. It returns all the elements up until that point. The index of each source element is provided as the second argument to `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1

Product	Versions
.NET Standard	2.0, 2.1
UWP	10.0

TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Returns elements from a sequence as long as a specified condition is true.

```
C#

public static System.Linq.IQueryable<TSource> TakeWhile<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

The sequence to return elements from.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements from the input sequence occurring before the element at which the test specified by `predicate` no longer passes.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use TakeWhile<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to return elements from the start of a sequence as long as a condition is true.

```C#
string[] fruits = { "apple", "banana", "mango", "orange",
                      "passionfruit", "grape" };

// Take strings from the array until a string
// that is equal to "orange" is found.
IEnumerable<string> query =
    fruits.AsQueryable()
    .TakeWhile(fruit => String.Compare("orange", fruit, true) != 0);

foreach (string fruit in query)
    Console.WriteLine(fruit);

/*
    This code produces the following output:

    apple
    banana
    mango
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling TakeWhile<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type

of the `source` parameter. The expected behavior is that it applies `predicate` to each element in `source` until it finds an element for which `predicate` returns `false`. It returns all the elements up until that point.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.ThenBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Performs a subsequent ordering of the elements in a sequence in ascending order.

Overloads

Expand table

ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.
ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.

C#

```
public static System.Linq.IOrderedQueryable<TSource> ThenBy<TSource,TKey>
(this System.Linq.IOrderedQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented by `keySelector`.

Parameters

`source` IOrderedQueryable<TSource>

An IOrderedQueryable<T> that contains elements to sort.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from each element.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Examples

The following code example demonstrates how to use ThenBy<TSource,TKey> (IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) to perform a secondary ordering of the elements in a sequence.

C#

```
string[] fruits = { "grape", "passionfruit", "banana", "apple",
                      "orange", "raspberry", "mango", "blueberry" };

// Sort the strings first by their length and then
// alphabetically by passing the identity selector function.
IEnumerable<string> query =
    fruits.AsQueryable()
    .OrderBy(fruit => fruit.Length).ThenBy(fruit => fruit);

foreach (string fruit in query)
    Console.WriteLine(fruit);
```

```
/*
    This code produces the following output:

    apple
    grape
    mango
    banana
    orange
    blueberry
    raspberry
    passionfruit
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) method generates a MethodCallExpression that represents calling ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it performs a secondary sort of the elements of `source` based on the key obtained by invoking `keySelector` on each element of `source`. All previously established sort orders are preserved.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9

Product	Versions
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)

Source: Queryable.cs

Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedQueryable<TSource> ThenBy<TSource,TKey>
(this System.Linq.IOrderedQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented by `keySelector`.

Parameters

`source` IOrderedQueryable<TSource>

An IOrderedQueryable<T> that contains elements to sort.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from each element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `comparer` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) method generates a MethodCallExpression that represents calling ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it performs a secondary sort of the elements of `source` based on the key obtained by invoking `keySelector` on each element of `source`. All previously established sort orders are preserved. The `comparer` parameter is used to compare key values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.ThenByDescending Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Performs a subsequent ordering of the elements in a sequence in descending order.

Overloads

Expand table

ThenByDescending<TSource,TKey> (IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)	Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.
ThenByDescending<TSource,TKey> (IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)	Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

ThenByDescending<TSource,TKey> (IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.

C#

```
public static System.Linq.IOrderedQueryable<TSource>
ThenByDescending<TSource,TKey> (this
System.Linq.IOrderedQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key returned by the function represented by `keySelector`.

Parameters

`source` IOrderedQueryable<TSource>

An IOrderedQueryable<T> that contains elements to sort.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from each element.

Returns

IOrderedQueryable<TSource>

An IOrderedQueryable<T> whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` is `null`.

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) method generates a MethodCallExpression that represents calling ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling

CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it performs a secondary sort of the elements of `source` in descending order, based on the key obtained by invoking `keySelector` on each element of `source`. All previously established sort orders are preserved.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>)

Source: Queryable.cs

Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

C#

```
public static System.Linq.IOrderedQueryable<TSource>
ThenByDescending<TSource,TKey> (this
System.Linq.IOrderedQueryable<TSource> source,
```

```
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of `source`.

`TKey`

The type of the key that is returned by the `keySelector` function.

Parameters

`source` IOrderedQueryable<TSource>

An IOrderedQueryable<T> that contains elements to sort.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract a key from each element.

`comparer` IComparer<TKey>

An IComparer<T> to compare keys.

Returns

IOrderedQueryable<TSource>

A collection whose elements are sorted in descending order according to a key.

Exceptions

ArgumentNullException

`source` or `keySelector` or `comparer` is `null`.

Examples

The following code example demonstrates how to use ThenByDescending<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) to perform a secondary ordering of the elements in a sequence in descending order by using a custom comparer.

C#

```
public class CaseInsensitiveComparer : IComparer<string>
{
    public int Compare(string x, string y)
    {
        return string.Compare(x, y, true);
    }
}

public static void ThenByDescendingEx1()
{
    string[] fruits =
    { "apPLe", "baNanA", "apple", "APple", "orange", "BAnana", "ORANGE",
"apPLE" };

    // Sort the strings first ascending by their length and
    // then descending using a custom case insensitive comparer.
    IEnumerable<string> query =
        fruits.AsQueryable()
        .OrderBy(fruit => fruit.Length)
        .ThenByDescending(fruit => fruit, new CaseInsensitiveComparer());

    foreach (string fruit in query)
        Console.WriteLine(fruit);
}

/*
    This code produces the following output:

    apPLe
    apple
    APple
    apPLE
    orange
    ORANGE
    baNanA
    BAnana
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) method generates a MethodCallExpression that represents calling ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>,

IComparer<TKey>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter. The result of calling CreateQuery<TElement>(Expression) is cast to type IOrderedQueryable<T> and returned.

The query behavior that occurs as a result of executing an expression tree that represents calling ThenBy<TSource,TKey>(IOrderedQueryable<TSource>, Expression<Func<TSource,TKey>>, IComparer<TKey>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it performs a secondary sort of the elements of `source` in descending order, based on the key obtained by invoking `keySelector` on each element of `source`. All previously established sort orders are preserved. The `comparer` parameter is used to compare key values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Union Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Produces the set union of two sequences.

Overloads

Expand table

Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>)	Produces the set union of two sequences by using the default equality comparer.
Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)	Produces the set union of two sequences by using a specified IEqualityComparer<T>.

Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>)

Source: Queryable.cs

Produces the set union of two sequences by using the default equality comparer.

C#

```
public static System.Linq.IQueryable<TSource> Union<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2);
```

Type Parameters

TSource

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

A sequence whose distinct elements form the first set for the union operation.

`source2` IEnumerable<TSource>

A sequence whose distinct elements form the second set for the union operation.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Examples

The following code example demonstrates how to use Union<TSource> (IQueryable<TSource>, IEnumerable<TSource>) to obtain the set union of two sequences.

C#

```
int[] ints1 = { 5, 3, 9, 7, 5, 9, 3, 7 };
int[] ints2 = { 8, 3, 6, 4, 4, 9, 1, 0 };

// Get the set union of the items in the two arrays.
IEnumerable<int> union = ints1.AsQueryable().Union(ints2);

foreach (int num in union)
    Console.Write("{0} ", num);

/*
    This code produces the following output:

    5 3 9 7 8 6 4 1 0
*/
```

Remarks

The Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>) method generates a MethodCallExpression that represents calling Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that the set union of the elements in `source1` and `source2` is returned.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>)

Source: Queryable.cs

Produces the set union of two sequences by using a specified IEqualityComparer<T>.

C#

```
public static System.Linq.IQueryable<TSource> Union<TSource> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2,
System.Collections.Generic.IEqualityComparer<TSource>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

Parameters

`source1` IQueryable<TSource>

A sequence whose distinct elements form the first set for the union operation.

`source2` IEnumerable<TSource>

A sequence whose distinct elements form the second set for the union operation.

`comparer` IEqualityComparer<TSource>

An IEqualityComparer<T> to compare values.

Returns

IQueryable<TSource>

An IQueryable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Remarks

The Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) method generates a MethodCallExpression that represents calling Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Union<TSource>(IQueryable<TSource>, IEnumerable<TSource>, IEqualityComparer<TSource>) depends on the implementation of the type of the `source1` parameter. The expected behavior is that the set union of the elements in

`source1` and `source2` is returned. The `comparer` parameter is used to compare values.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.UnionBy Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>)	Produces the set union of two sequences according to a specified key selector function.
UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)	Produces the set union of two sequences according to a specified key selector function.

UnionBy<TSource,TKey>(IQueryable<TSource>, IEnumerable<TSource>, Expression<Func<TSource,TKey>>)

Source: Queryable.cs

Produces the set union of two sequences according to a specified key selector function.

C#

```
public static System.Linq.IQueryable<TSource> UnionBy<TSource,TKey> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose distinct elements form the first set for the union.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the second set for the union.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

Returns

IQueryable<TSource>

An IEnumerable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

UnionBy<TSource,TKey> (IQueryable<TSource>, IEnumerable<TSource>,

Expression<Func<TSource,TKey>>, IEqualityComparer<TKey>)

Source: Queryable.cs

Produces the set union of two sequences according to a specified key selector function.

```
C#

public static System.Linq.IQueryable<TSource> UnionBy<TSource,TKey> (this
System.Linq.IQueryable<TSource> source1,
System.Collections.Generic.IEnumerable<TSource> source2,
System.Linq.Expressions.Expression<Func<TSource,TKey>> keySelector,
System.Collections.Generic.IEqualityComparer<TKey>? comparer);
```

Type Parameters

`TSource`

The type of the elements of the input sequences.

`TKey`

The type of key to identify elements by.

Parameters

`source1` IQueryable<TSource>

An IQueryable<T> whose distinct elements form the first set for the union.

`source2` IEnumerable<TSource>

An IEnumerable<T> whose distinct elements form the second set for the union.

`keySelector` Expression<Func<TSource,TKey>>

A function to extract the key for each element.

`comparer` IEqualityComparer<TKey>

The IEqualityComparer<T> to compare values.

Returns

IQueryable<TSource>

An IEnumerable<T> that contains the elements from both input sequences, excluding duplicates.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Queryable.Where Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Filters a sequence of values based on a predicate.

Overloads

Expand table

Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)	Filters a sequence of values based on a predicate.
Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)	Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>)

Source: Queryable.cs

Filters a sequence of values based on a predicate.

C#

```
public static System.Linq.IQueryable<TSource> Where<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,bool>> predicate);
```

Type Parameters

TSource

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to filter.

`predicate` Expression<Func<TSource,Boolean>>

A function to test each element for a condition.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements from the input sequence that satisfy the condition specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use Where<TSource> (IQueryable<TSource>, Expression<Func<TSource,Boolean>>) to filter a sequence.

C#

```
List<string> fruits =
    new List<string> { "apple", "passionfruit", "banana", "mango",
                       "orange", "blueberry", "grape", "strawberry" };

// Get all strings whose length is less than 6.
IEnumerable<string> query =
    fruits.AsQueryable().Where(fruit => fruit.Length < 6);

foreach (string fruit in query)
    Console.WriteLine(fruit);

/*
    This code produces the following output:

    apple
    mango
    grape
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) method generates a MethodCallExpression that represents calling Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the elements from `source` that satisfy the condition specified by `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>)

Source: Queryable.cs

Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

```C#
public static System.Linq.IQueryable<TSource> Where<TSource> (this
System.Linq.IQueryable<TSource> source,
System.Linq.Expressions.Expression<Func<TSource,int,bool>> predicate);
```

Type Parameters

`TSource`

The type of the elements of `source`.

Parameters

`source` IQueryable<TSource>

An IQueryable<T> to filter.

`predicate` Expression<Func<TSource,Int32,Boolean>>

A function to test each element for a condition; the second parameter of the function represents the index of the element in the source sequence.

Returns

IQueryable<TSource>

An IQueryable<T> that contains elements from the input sequence that satisfy the condition specified by `predicate`.

Exceptions

ArgumentNullException

`source` or `predicate` is `null`.

Examples

The following code example demonstrates how to use Where<TSource> (IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) to filter a sequence based on a predicate that incorporates the index of each element.

```C#
int[] numbers = { 0, 30, 20, 15, 90, 85, 40, 75 };

// Get all the numbers that are less than or equal to
```

```
// the product of their index in the array and 10.
IEnumerable<int> query =
    numbers.AsQueryable()
    .Where((number, index) => number <= index * 10);

foreach (int number in query)
    Console.WriteLine(number);

/*
    This code produces the following output:

    0
    20
    15
    40
*/
```

Remarks

This method has at least one parameter of type Expression<TDelegate> whose type argument is one of the Func<T,TResult> types. For these parameters, you can pass in a lambda expression and it will be compiled to an Expression<TDelegate>.

The Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) method generates a MethodCallExpression that represents calling Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery(Expression) method of the IQueryProvider represented by the Provider property of the `source` parameter.

The query behavior that occurs as a result of executing an expression tree that represents calling Where<TSource>(IQueryable<TSource>, Expression<Func<TSource,Int32,Boolean>>) depends on the implementation of the type of the `source` parameter. The expected behavior is that it returns the elements from `source` that satisfy the condition specified by `predicate`. The index of each source element is provided as the second argument to `predicate`.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Queryable.Zip Method

Reference

Definition

Namespace: System.Linq

Assembly: System.Linq.Queryable.dll

Overloads

Expand table

Zip<TFirst,TSecond,TResult>(IQueryable<TFirst>, IEnumerable<TSecond>, Expression<Func<TFirst,TSecond,TResult>>)	Merges two sequences by using the specified predicate function.
Zip<TFirst,TSecond,TThird>(IQueryable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from the three specified sequences.
Zip<TFirst,TSecond>(IQueryable<TFirst>, IEnumerable<TSecond>)	Produces a sequence of tuples with elements from the two specified sequences.

Zip<TFirst,TSecond,TResult>(IQueryable<TFirst>, IEnumerable<TSecond>, Expression<Func<TFirst,TSecond,TResult>>)

Source: Queryable.cs

Merges two sequences by using the specified predicate function.

C#

```
public static System.Linq.IQueryable<TResult> Zip<TFirst,TSecond,TResult>
(this System.Linq.IQueryable<TFirst> source1,
System.Collections.Generic.IEnumerable<TSecond> source2,
System.Linq.Expressions.Expression<Func<TFirst,TSecond,TResult>>
resultSelector);
```

Type Parameters

`TFirst`

The type of the elements of the first input sequence.

`TSecond`

The type of the elements of the second input sequence.

`TResult`

The type of the elements of the result sequence.

Parameters

`source1` IQueryable<TFirst>

The first sequence to merge.

`source2` IEnumerable<TSecond>

The second sequence to merge.

`resultSelector` Expression<Func<TFirst,TSecond,TResult>>

A function that specifies how to merge the elements from the two sequences.

Returns

IQueryable<TResult>

An IQueryable<T> that contains merged elements of two input sequences.

Exceptions

ArgumentNullException

`source1` or `source2` is `null`.

Examples

The following code example demonstrates how to use the Zip method to merge two sequences.

C#

```
int[] numbers = { 1, 2, 3, 4 };
string[] words = { "one", "two", "three" };
```

```
var numbersAndWords = numbers.AsQueryable().Zip(words, (first, second) =>
first + " " + second);

foreach (var item in numbersAndWords)
    Console.WriteLine(item);

// This code produces the following output:

// 1 one
// 2 two
// 3 three
```

Remarks

The Zip method generates a MethodCallExpression that represents calling Zip itself as a constructed generic method. It then passes the MethodCallExpression to the CreateQuery<TElement>(Expression) method of the IQueryProvider represented by the Provider property of the `source1` parameter.

The method merges each element of the first sequence with an element that has the same index in the second sequence. If the sequences do not have the same number of elements, the method merges sequences until it reaches the end of one of them. For example, if one sequence has three elements and the other one has four, the resulting sequence will have only three elements.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8, 9
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	2.0, 2.1
UWP	10.0

Zip<TFirst,TSecond,TThird>(IQueryable<TFirst>, IEnumerable<TSecond>,

IEnumerable<TThird>)

Source: Queryable.cs

Produces a sequence of tuples with elements from the three specified sequences.

```
C#

public static System.Linq.IQueryable<(TFirst First, TSecond Second,
TThird Third)> Zip<TFirst,TSecond,TThird> (this
System.Linq.IQueryable<TFirst> source1,
System.Collections.Generic.IEnumerable<TSecond> source2,
System.Collections.Generic.IEnumerable<TThird> source3);
```

Type Parameters

`TFirst`

The type of the elements of the first input sequence.

`TSecond`

The type of the elements of the second input sequence.

`TThird`

The type of the elements of the third input sequence.

Parameters

`source1` IQueryable<TFirst>

The first sequence to merge.

`source2` IEnumerable<TSecond>

The second sequence to merge.

`source3` IEnumerable<TThird>

The third sequence to merge.

Returns

IQueryable<ValueTuple<TFirst,TSecond,TThird>>

A sequence of tuples with elements taken from the first, second and third sequences, in that order.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	6, 7, 8, 9

Zip<TFirst,TSecond>(IQueryable<TFirst>, IEnumerable<TSecond>)

Source: Queryable.cs

Produces a sequence of tuples with elements from the two specified sequences.

C#

```
public static System.Linq.IQueryable<(TFirst First, TSecond Second)>
Zip<TFirst,TSecond> (this System.Linq.IQueryable<TFirst> source1,
System.Collections.Generic.IEnumerable<TSecond> source2);
```

Type Parameters

`TFirst`

The type of the elements of the first input sequence.

`TSecond`

The type of the elements of the second input sequence.

Parameters

`source1` IQueryable<TFirst>

The first sequence to merge.

`source2` IEnumerable<TSecond>

The second sequence to merge.

Returns

IQueryable<ValueTuple<TFirst,TSecond>>

A sequence of tuples with elements taken from the first and second sequences, in that order.

Applies to

▼ .NET 9 and other versions

Product	Versions
.NET	Core 3.0, Core 3.1, 5, 6, 7, 8, 9