Queue<T> Class

Namespace: System.Collections.Generic

Assemblies: System.Collections.dll, System.dll, netstandard.dll

Represents a first-in, first-out collection of objects.

In this article

C# Copy

```csharp
[System.Runtime.InteropServices.ComVisible(false)]
[System.Serializable]
public class Queue<T> : System.Collections.Generic.IEnumerable<T>,
System.Collections.Generic.IReadOnlyCollection<T>,
System.Collections.ICollection
```

Type Parameters

T

Specifies the type of elements in the queue.

Inheritance Object → Queue<T>

Attributes ComVisibleAttribute, SerializableAttribute

Implements IEnumerable<T>, IReadOnlyCollection<T>, ICollection, IEnumerable

Examples

The following code example demonstrates several methods of the Queue<T> generic class. The code example creates a queue of strings with default capacity and uses the Enqueue method to queue five strings. The elements of the queue are enumerated, which does not change the state of the queue. The Dequeue method is used to dequeue the first string. The Peek method is used to look at the next item in the queue, and then the Dequeue method is used to dequeue it.

The ToArray method is used to create an array and copy the queue elements to it, then the array is passed to the Queue<T> constructor that takes IEnumerable<T>, creating a copy of the queue. The elements of the copy are displayed.

An array twice the size of the queue is created, and the CopyTo method is used to copy the array elements beginning at the middle of the array. The Queue<T> constructor is used again to create a second copy of the queue containing three null elements at the beginning.

The Contains method is used to show that the string "four" is in the first copy of the queue, after which the Clear method clears the copy and the Count property shows that the queue is empty.

C# Copy

```csharp
using System;
using System.Collections.Generic;

class Example
{
    public static void Main()
    {
        Queue<string> numbers = new Queue<string>();
        numbers.Enqueue("one");
        numbers.Enqueue("two");
        numbers.Enqueue("three");
        numbers.Enqueue("four");
        numbers.Enqueue("five");

        // A queue can be enumerated without disturbing its contents.
        foreach( string number in numbers )
        {
            Console.WriteLine(number);
        }

        Console.WriteLine("\nDequeuing '{0}'", numbers.Dequeue());
        Console.WriteLine("Peek at next item to dequeue: {0}",
            numbers.Peek());
        Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());

        // Create a copy of the queue, using the ToArray method and the
        // constructor that accepts an IEnumerable<T>.
        Queue<string> queueCopy = new Queue<string>(numbers.ToArray());

        Console.WriteLine("\nContents of the first copy:");
        foreach( string number in queueCopy )
        {
            Console.WriteLine(number);
        }

        // Create an array twice the size of the queue and copy the
        // elements of the queue, starting at the middle of the
        // array.
        string[] array2 = new string[numbers.Count * 2];
        numbers.CopyTo(array2, numbers.Count);

        // Create a second queue, using the constructor that accepts an
        // IEnumerable(Of T).
        Queue<string> queueCopy2 = new Queue<string>(array2);

        Console.WriteLine("\nContents of the second copy, with duplicates and
nulls:");
        foreach( string number in queueCopy2 )
        {
            Console.WriteLine(number);
        }

        Console.WriteLine("\nqueueCopy.Contains(\"four\") = {0}",
            queueCopy.Contains("four"));

        Console.WriteLine("\nqueueCopy.Clear()");
        queueCopy.Clear();
        Console.WriteLine("\nqueueCopy.Count = {0}", queueCopy.Count);
    }
}

/* This code example produces the following output:

one
two
three
four
five

Dequeuing 'one'
Peek at next item to dequeue: two
Dequeuing 'two'
```

```
Contents of the copy:
three
four
five

Contents of the second copy, with duplicates and nulls:



three
four
five

queueCopy.Contains("four") = True

queueCopy.Clear()

queueCopy.Count = 0
 */
```

Remarks

This class implements a generic queue as a circular array. Objects stored in a Queue<T> are inserted at one end and removed from the other. Queues and stacks are useful when you need temporary storage for information; that is, when you might want to discard an element after retrieving its value. Use Queue<T> if you need to access the information in the same order that it is stored in the collection. Use Stack<T> if you need to access the information in reverse order. Use ConcurrentQueue<T> or ConcurrentStack<T> if you need to access the collection from multiple threads concurrently.

Three main operations can be performed on a Queue<T> and its elements:

- Enqueue adds an element to the end of the Queue<T>.
- Dequeue removes the oldest element from the start of the Queue<T>.
- Peek peek returns the oldest element that is at the start of the Queue<T> but does not remove it from the Queue<T>.

The capacity of a Queue<T> is the number of elements the Queue<T> can hold. As elements are added to a Queue<T>, the capacity is automatically increased as required by reallocating the internal array. The capacity can be decreased by calling TrimExcess.

Queue<T> accepts `null` as a valid value for reference types and allows duplicate elements.

Constructors

Queue<T>()	Initializes a new instance of the Queue<T> class that is empty and has the default initial capacity.
Queue<T>(IEnumerable<T>)	Initializes a new instance of the Queue<T> class that contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.
Queue<T>(Int32)	Initializes a new instance of the Queue<T> class that is empty and has the specified initial capacity.

Properties

Count	Gets the number of elements contained in the Queue<T>.

Methods

Clear()	Removes all objects from the Queue<T>.
Contains(T)	Determines whether an element is in the Queue<T>.
CopyTo(T[], Int32)	Copies the Queue<T> elements to an existing one-dimensional Array, starting at the specified array index.
Dequeue()	Removes and returns the object at the beginning of the Queue<T>.
Enqueue(T)	Adds an object to the end of the Queue<T>.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
GetEnumerator()	Returns an enumerator that iterates through the Queue<T>.

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)
Peek()	Returns the object at the beginning of the Queue<T> without removing it.
ToArray()	Copies the Queue<T> elements to a new array.
ToString()	Returns a string that represents the current object. (Inherited from Object)
TrimExcess()	Sets the capacity to the actual number of elements in the Queue<T>, if that number is less than 90 percent of current capacity.

Explicit Interface Implementations

ICollection.CopyTo(Array, Int32)	Copies the elements of the ICollection to an Array, starting at a particular Array index.
ICollection.IsSynchronized	Gets a value indicating whether access to the ICollection is synchronized (thread safe).
ICollection.SyncRoot	Gets an object that can be used to synchronize access to the ICollection.
IEnumerable.GetEnumerator()	Returns an enumerator that iterates through a collection.
IEnumerable<T>.GetEnumerator()	Returns an enumerator that iterates through a collection.

Extension Methods

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>,	Determines whether any element of a sequence satisfies a condition.

Func<TSource,Boolean>)	
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>,	Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input

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

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.
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.
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>,	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.

Func<TSource,TKey>, Func<TSource,TElement>)	
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.
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>,	Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer<T> is used to compare keys.

Func<TInner,TKey>, Func<TOuter,TInner,TResult>, IEqualityComparer<TKey>)	
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.
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>(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,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.
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>(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,TResult>(IEnumerable<TSource>, Func<TSource,TResult>)	Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.
OfType<TResult>(IEnumerable)	Filters the elements of an IEnumerable based on a specified type.
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.
Prepend<TSource>	Adds a value to the beginning of the sequence.

(IEnumerable<TSource>, TSource)	
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>, 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.
Skip<TSource>(IEnumerable<TSource>, Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining elements.
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>	Computes the sum of the sequence of nullable Decimal values that are

(IEnumerable<TSource>, Func<TSource,Nullable<Decimal>>)	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.
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>	Creates a Lookup<TKey,TElement> from an IEnumerable<T> according

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

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.
AncestorsAndSelf(IEnumerable<XElement>)	Returns a collection of elements that contains every element in the source collection, and the ancestors of every element in the source collection.
AncestorsAndSelf(IEnumerable<XElement>, XName)	Returns a filtered collection of elements that contains every element in the source collection, and the ancestors of every element in the source collection. Only elements that have a matching XName are included in the collection.
Attributes(IEnumerable<XElement>)	Returns a collection of the attributes of every element in the source collection.
Attributes(IEnumerable<XElement>, XName)	Returns a filtered collection of the attributes of every element 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.
DescendantNodesAndSelf(IEnumerable<XElement>)	Returns a collection of nodes that contains every element in the source collection, and the descendant nodes of every element in the source collection.
Descendants<T>	Returns a collection of elements that contains the descendant elements

(IEnumerable<T>)	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.
DescendantsAndSelf(IEnumerable<XElement>)	Returns a collection of elements that contains every element in the source collection, and the descendent elements of every element in the source collection.
DescendantsAndSelf(IEnumerable<XElement>, XName)	Returns a filtered collection of elements that contains every element in the source collection, and the descendents of every element 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(IEnumerable<XAttribute>)	Removes every attribute in the source collection from its parent element.
Remove<T>(IEnumerable<T>)	Removes every node in the source collection from its parent node.

Applies to

.NET Core

3.0 Preview 2, 2.2, 2.1, 2.0, 1.1, 1.0

.NET Framework

4.8, 4.7.2, 4.7.1, 4.7, 4.6.2, 4.6.1, 4.6, 4.5.2, 4.5.1, 4.5, 4.0, 3.5, 3.0, 2.0

.NET Standard

2.0, 1.6, 1.4, 1.3, 1.2, 1.1, 1.0

UWP

10.0

Xamarin.Android

7.1

Xamarin.iOS

10.8

Xamarin.Mac

3.0

Thread Safety

Public static (`Shared` in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

A Queue<T> can support multiple readers concurrently, as long as the collection is not modified. Even so, enumerating through a collection is intrinsically not a thread-safe procedure. To guarantee thread safety during enumeration, you can lock the collection during the entire enumeration. To allow the collection to be accessed by multiple threads for reading and writing, you must implement your own synchronization.