

# 1 System.Collections.Generic.Queue<T> Class

```
2 [ILAsm]
3 .class public serializable beforefieldinit Queue`1<T> extends
4 System.Object implements System.Collections.Generic.IEnumerable`1<!0>,
5 System.Collections.ICollection, System.Collections.IEnumerable
6
7 [C#]
8 public class Queue<T>: System.Collections.Generic.IEnumerable<T>,
9 System.Collections.ICollection
```

## 9 Assembly Info:

- 10 • *Name:* System
- 11 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- 12 • *Version:* 4.0.0.0
- 13 • *Attributes:*
  - 14 ○ CLSCompliantAttribute(true)

## 15 Implements:

- 16 • **System.Collections.Generic.IEnumerable<T>**
- 17 • **System.Collections.ICollection**

## 18 Summary

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

## 20 Inherits From: System.Object

21  
22 **Library:** BCL

## 24 Description

25 Queues are useful for storing messages in the order they were received for sequential  
26 processing. Objects stored in a `System.Collections.Generic.Queue`1<T>` are inserted  
27 at one end and removed from the other.

28  
29 The capacity of a `System.Collections.Generic.Queue`1<T>` is the number of elements  
30 the `System.Collections.Generic.Queue`1<T>` can hold. As elements are added to a  
31 `System.Collections.Generic.Queue`1<T>`, the capacity is automatically increased as  
32 required by reallocating the internal array. The capacity can be decreased by calling  
33 `System.Collections.Generic.Queue`1<T>.TrimExcess`.

34  
35 `System.Collections.Generic.Queue`1<T>` accepts null as a valid value for reference  
36 types and allows duplicate elements.

37

# 1 Queue<T>() Constructor

```
2 [ILAsm]  
3 .method public hidebysig specialname rtspecialname instance void .ctor()  
4 cil managed  
  
5 [C#]  
6 public Queue ()
```

## 7 Summary

8 Initializes a new instance of the `System.Collections.Generic.Queue`1<T>` class that is  
9 empty and has the default initial capacity.

## 10 Description

11 The capacity of a `System.Collections.Generic.Queue`1<T>` is the number of elements  
12 that the `System.Collections.Generic.Queue`1<T>` can hold. As elements are added to  
13 a `System.Collections.Generic.Queue`1<T>`, the capacity is automatically increased as  
14 required by reallocating the internal array.

15  
16 If the size of the collection can be estimated, specifying the initial capacity eliminates  
17 the need to perform a number of resizing operations while adding elements to the  
18 `System.Collections.Generic.Queue`1<T>`.

19  
20 The capacity can be decreased by calling  
21 `System.Collections.Generic.Queue`1<T>.TrimExcess`.

22  
23 This constructor is an O(1) operation.

24

# Queue<T> (System.Collections.Generic.IEnumerable<T>) Constructor

```
[ILAsm]
.method public hidebysig specialname rtspecialname instance void
.ctor(class System.Collections.Generic.IEnumerable`1<!0> collection) cil
managed

[C#]
public Queue (System.Collections.Generic.IEnumerable<T> collection)
```

## Summary

Initializes a new instance of the `System.Collections.Generic.Queue`1<T>` class that contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.

## Parameters

Parameter	Description
<i>collection</i>	The collection whose elements are copied to the new <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> .

## Description

The capacity of a `System.Collections.Generic.Queue`1<T>` is the number of elements that the `System.Collections.Generic.Queue`1<T>` can hold. As elements are added to a `System.Collections.Generic.Queue`1<T>`, the capacity is automatically increased as required by reallocating the internal array.

If the size of the collection can be estimated, specifying the initial capacity eliminates the need to perform a number of resizing operations while adding elements to the `System.Collections.Generic.Queue`1<T>`.

The capacity can be decreased by calling `System.Collections.Generic.Queue`1<T>.TrimExcess`.

The elements are copied onto the `System.Collections.Generic.Queue`1<T>` in the same order they are read by the `System.Collections.Generic.IEnumerator`1<T>` of the collection.

This constructor is an  $O(n)$  operation, where  $n$  is the number of elements in *collection*.

## Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>collection</i> is null.

1

2

# Queue<T> (System.Int32) Constructor

```
[ILAsm]
.method public hidebysig specialname rtspecialname instance void
.ctor(int32 capacity) cil managed

[C#]
public Queue (int capacity)
```

## Summary

Initializes a new instance of the `System.Collections.Generic.Queue`1<T>` class that is empty and has the specified initial capacity.

## Parameters

Parameter	Description
<i>capacity</i>	The initial number of elements that the <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> can contain.

## Description

The capacity of a `System.Collections.Generic.Queue`1<T>` is the number of elements that the `System.Collections.Generic.Queue`1<T>` can hold. As elements are added to a `System.Collections.Generic.Queue`1<T>`, the capacity is automatically increased as required by reallocating the internal array.

If the size of the collection can be estimated, specifying the initial capacity eliminates the need to perform a number of resizing operations while adding elements to the `System.Collections.Generic.Queue`1<T>`.

The capacity can be decreased by calling `System.Collections.Generic.Queue`1<T>.TrimExcess`.

This constructor is an  $O(n)$  operation, where  $n$  is *capacity*.

## Exceptions

Exception	Condition
<code>System.ArgumentOutOfRangeException</code>	<i>capacity</i> is less than zero.

# 1 Queue<T>.Clear() Method

```
2 [ILAsm]  
3 .method public hidebysig instance void Clear() cil managed  
4 [C#]  
5 public void Clear ()
```

## 6 Summary

7 Removes all objects from the `System.Collections.Generic.Queue`1<T>`.

## 8 Description

9 `System.Collections.Generic.Queue`1<T>.Count` is set to zero, and references to  
10 other objects from elements of the collection are also released.

11  
12 The capacity remains unchanged. To reset the capacity of the  
13 `System.Collections.Generic.Queue`1<T>`, call  
14 `System.Collections.Generic.Queue`1<T>.TrimExcess`. Trimming an empty  
15 `System.Collections.Generic.Queue`1<T>` sets the capacity of the  
16 `System.Collections.Generic.Queue`1<T>` to the default capacity.

17  
18 This method is an  $O(n)$  operation, where  $n$  is  
19 `System.Collections.Generic.Queue`1<T>.Count`.

20

# 1 Queue<T>.Contains(T) Method

```
2 [ILAsm]  
3 .method public hidebysig instance bool Contains(!0 item) cil managed  
4 [C#]  
5 public bool Contains (T item)
```

## 6 Summary

7 Determines whether an element is in the `System.Collections.Generic.Queue`1<T>`.

## 8 Parameters

Parameter	Description
<i>item</i>	The object to locate in the <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> . The value can be null for reference types.

## 9 Return Value

11 true if *item* is found in the `System.Collections.Generic.Queue`1<T>`; otherwise,  
12 false.

## 13 Description

14 This method determines equality using the default equality comparer  
15 `System.Collections.Generic.EqualityComparer`1<T>.Default` for *T*, the type of  
16 values in the queue.

17  
18 This method performs a linear search; therefore, this method is an  $O(n)$  operation,  
19 where *n* is `System.Collections.Generic.Queue`1<T>.Count`.

20

# 1 Queue<T>.CopyTo(T[], System.Int32)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance void CopyTo(!0[] array, int32  
5 arrayIndex) cil managed  
  
6 [C#]  
7 public void CopyTo (T[] array, int arrayIndex)
```

### 8 Summary

9 Copies the `System.Collections.Generic.Queue`1<T>` elements to an existing one-  
10 dimensional `System.Array`, starting at the specified array index.

### 11 Parameters

Parameter	Description
<i>array</i>	The one-dimensional <code>System.Array</code> that is the destination of the elements copied from <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> . The <code>System.Array</code> must have zero-based indexing.
<i>arrayIndex</i>	The zero-based index in <i>array</i> at which copying begins.

### 12 13 Description

14 The elements are copied to the `System.Array` in the same order in which the  
15 enumerator iterates through the `System.Collections.Generic.Queue`1<T>`.

16 This method is an  $O(n)$  operation, where  $n$  is  
17 `System.Collections.Generic.Queue`1<T>.Count`.  
18

### 19 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>array</i> is null.
<code>System.ArgumentOutOfRangeException</code>	<i>arrayIndex</i> is less than zero.
<code>System.ArgumentException</code>	The number of elements in the source <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> is greater than the available space from

	<i>arrayIndex</i> to the end of the destination <i>array</i> .
--	--

1

2

# 1 Queue<T>.Dequeue() Method

```
2 [ILAsm]  
3 .method public hidebysig instance !0 Dequeue() cil managed  
4 [C#]  
5 public T Dequeue ()
```

## 6 Summary

7 Removes and returns the object at the beginning of the  
8 System.Collections.Generic.Queue`1<T>.

## 9 Return Value

10 The object that is removed from the beginning of the  
11 System.Collections.Generic.Queue`1<T>.

## 12 Description

13 This method is similar to the System.Collections.Generic.Queue`1<T>.Peek method,  
14 but System.Collections.Generic.Queue`1<T>.Peek does not modify the  
15 System.Collections.Generic.Queue`1<T>.

16  
17 If type *T* is a reference type, null can be added to the  
18 System.Collections.Generic.Queue`1<T> as a value.

19  
20 This method is an O(1) operation.

## 21 Exceptions

Exception	Condition
System.InvalidOperationException	The System.Collections.Generic.Queue`1<T> is empty.

22

23

# 1 Queue<T>.Enqueue(T) Method

```
2 [ILAsm]  
3 .method public hidebysig instance void Enqueue(!0 item) cil managed  
4 [C#]  
5 public void Enqueue (T item)
```

## 6 Summary

7 Adds an object to the end of the `System.Collections.Generic.Queue`1<T>`.

## 8 Parameters

Parameter	Description
<i>item</i>	The object to add to the <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> . The value can be null for reference types.

9

## 10 Description

11 If `System.Collections.Generic.Queue`1<T>.Count` already equals the capacity, the  
12 capacity of the `System.Collections.Generic.Queue`1<T>` is increased by automatically  
13 reallocating the internal array, and the existing elements are copied to the new array  
14 before the new element is added.

15

16 If `System.Collections.Generic.Queue`1<T>.Count` is less than the capacity of the  
17 internal array, this method is an  $O(1)$  operation. If the internal array needs to be  
18 reallocated to accommodate the new element, this method becomes an  $O(n)$  operation,  
19 where  $n$  is `System.Collections.Generic.Queue`1<T>.Count`.

20

# 1 Queue<T>.GetEnumerator() Method

```
2 [ILAsm]  
3 .method public hidebysig instance valuetype  
4 System.Collections.Generic.Queue`1/Enumerator<!0> GetEnumerator() cil  
5 managed  
  
6 [C#]  
7 public System.Collections.Generic.Queue<T>.Enumerator GetEnumerator ()
```

## 8 Summary

9 Returns an enumerator that iterates through the  
10 System.Collections.Generic.Queue`1<T>.

## 11 Return Value

12 An System.Collections.Generic.Queue`1<T>.Enumerator for the  
13 System.Collections.Generic.Queue`1<T>.

## 14 Description

## 15 Usage

16 For a detailed description regarding the use of an enumerator, see  
17 System.Collections.Generic.IEnumerator<T>.

18  
19  
20 Default implementations of collections in System.Collections.Generic are not  
21 synchronized.

22  
23 This method is an O(1) operation.

24

# 1 Queue<T>.Peek() Method

```
2 [ILAsm]  
3 .method public hidebysig instance !0 Peek() cil managed  
4 [C#]  
5 public T Peek ()
```

## 6 Summary

7 Returns the object at the beginning of the `System.Collections.Generic.Queue`1<T>`  
8 without removing it.

## 9 Return Value

10 The object at the beginning of the `System.Collections.Generic.Queue`1<T>`.

## 11 Description

12 This method is similar to the `System.Collections.Generic.Queue`1<T>.Dequeue`  
13 `method`, but `System.Collections.Generic.Queue`1<T>.Peek` does not modify the  
14 `System.Collections.Generic.Queue`1<T>`.

15  
16 If type *T* is a reference type, null can be added to the  
17 `System.Collections.Generic.Queue`1<T>` as a value.

18  
19 This method is an O(1) operation.

## 20 Exceptions

Exception	Condition
<code>System.InvalidOperationException</code>	The <code>System.Collections.Generic.Queue`1&lt;T&gt;</code> is empty.

21

22

# 1 2 Queue<T>.System.Collections.Generic.IEnum 3 erable<T>.GetEnumerator() Method

```
4 [ILAsm]  
5 .method private hidebysig newslot virtual final instance class  
6 System.Collections.Generic.IEnumerator`1<!0>  
7 System.Collections.Generic.IEnumerable<T>.GetEnumerator() cil managed  
8 [C#]  
9 System.Collections.Generic.IEnumerator<T> IEnumerable<T>.GetEnumerator ()
```

## 10 Summary

11 Returns an enumerator that iterates through a collection.

## 12 Return Value

13 An System.Collections.Generic.IEnumerator`1<T> that can be used to iterate  
14 through the collection.

## 15 Description

## 16 Usage

17 For a detailed description regarding the use of an enumerator, see  
18 System.Collections.Generic.IEnumerator<T>.

19  
20  
21 Default implementations of collections in System.Collections.Generic are not  
22 synchronized.

23  
24 This method is an O(1) operation.

25

1  
2 **Queue<T>.System.Collections.ICollection.CopyTo(System.Array, System.Int32) Method**  
3

```
4 [ILAsm]  
5 .method private hidebysig newslot virtual final instance void  
6 System.Collections.ICollection.CopyTo(class System.Array array, int32  
7 index) cil managed  
  
8 [C#]  
9 void ICollection.CopyTo (Array array, int index)
```

10 **Summary**

11 Copies the elements of the `System.Collections.ICollection` to an `System.Array`,  
12 starting at a particular `System.Array` index.

13 **Parameters**

Parameter	Description
<i>array</i>	The one-dimensional <code>System.Array</code> that is the destination of the elements copied from <code>System.Collections.ICollection</code> . The <code>System.Array</code> must have zero-based indexing.
<i>index</i>	The zero-based index in <i>array</i> at which copying begins.

14  
15 **Description**

16 [Note: If the type of the source `System.Collections.ICollection` cannot be cast  
17 automatically to the type of the destination *array*, the non-generic implementations of  
18 `System.Collections.ICollection.CopyTo` throw `System.InvalidCastException`,  
19 whereas the generic implementations throw `System.ArgumentException`.  
20

21 ]  
22  
23 This method is an  $O(n)$  operation, where  $n$  is  
24 `System.Collections.Generic.Queue<T>.Count`.

25 **Exceptions**

Exception	Condition
<code>System.ArgumentNullException</code>	<i>array</i> is null.

<b>System.ArgumentOutOfRangeException</b>	<i>index</i> is less than zero.
<b>System.ArgumentException</b>	<p><i>array</i> is multidimensional.</p> <p>-or-</p> <p><i>array</i> does not have zero-based indexing.</p> <p>-or-</p> <p>The number of elements in the source <code>System.Collections.ICollection</code> is greater than the available space from <i>index</i> to the end of the destination <i>array</i>.</p> <p>-or-</p> <p>The type of the source <code>System.Collections.ICollection</code> cannot be cast automatically to the type of the destination <i>array</i>.</p>

1

2

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

# Queue<T>.System.Collections.IEnumerable.GetEnumerator() Method

```
[ILAsm]
.method private hidebysig newslot virtual final instance class
System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator() cil managed

[C#]
System.Collections.IEnumerator IEnumerable.GetEnumerator ()
```

## Summary

Returns an enumerator that iterates through a collection.

## Return Value

An `System.Collections.IEnumerator` that can be used to iterate through the collection.

## Description

## Usage

For a detailed description regarding the use of an enumerator, see `System.Collections.Generic.IEnumerator<T>`.

Default implementations of collections in `System.Collections.Generic` are not synchronized.

This method is an O(1) operation.

# 1 Queue<T>.ToArray() Method

```
2 [ILAsm]  
3 .method public hidebysig instance !0[] ToArray() cil managed  
4 [C#]  
5 public T[] ToArray ()
```

## 6 Summary

7 Copies the `System.Collections.Generic.Queue<T>` elements to a new array.

## 8 Return Value

9 A new array containing elements copied from the  
10 `System.Collections.Generic.Queue<T>`.

## 11 Description

12 The `System.Collections.Generic.Queue<T>` is not modified. The order of the  
13 elements in the new array is the same as the order of the elements from the beginning  
14 of the `System.Collections.Generic.Queue<T>` to its end.

15  
16 This method is an  $O(n)$  operation, where  $n$  is  
17 `System.Collections.Generic.Queue<T>.Count`.

18

# 1 Queue<T>.TrimExcess() Method

```
2 [ILAsm]  
3 .method public hidebysig instance void TrimExcess() cil managed  
4 [C#]  
5 public void TrimExcess ()
```

## 6 Summary

7 Sets the capacity to the actual number of elements in the  
8 System.Collections.Generic.Queue`1<T>, if that number is less than 90 percent of  
9 current capacity.

## 10 Description

11 This method can be used to minimize a collection's memory overhead if no new  
12 elements will be added to the collection. The cost of reallocating and copying a large  
13 System.Collections.Generic.Queue`1<T> can be considerable, however, so the  
14 System.Collections.Generic.Queue`1<T>.TrimExcess method does nothing if the list  
15 is at more than 90 percent of capacity. This avoids incurring a large reallocation cost for  
16 a relatively small gain.

17 This method is an  $O(n)$  operation, where  $n$  is  
18 System.Collections.Generic.Queue`1<T>.Count.

19 To reset a System.Collections.Generic.Queue`1<T> to its initial state, call the  
20 System.Collections.Generic.Queue`1<T>.Clear method before calling  
21 System.Collections.Generic.Queue`1<T>.TrimExcess method. Trimming an empty  
22 System.Collections.Generic.Queue`1<T> sets the capacity of the  
23 System.Collections.Generic.Queue`1<T> to the default capacity.  
24  
25

26

# 1 Queue<T>.Count Property

```
2 [ILAsm]  
3 .property instance int32 Count  
4 [C#]  
5 public int Count { get; }
```

## 6 Summary

7 Gets the number of elements contained in the  
8 System.Collections.Generic.Queue`1<T>.

## 9 Property Value

10 The number of elements contained in the System.Collections.Generic.Queue`1<T>.

## 11 Description

12 The capacity of a System.Collections.Generic.Queue`1<T> is the number of elements  
13 that the System.Collections.Generic.Queue`1<T> can store.  
14 System.Collections.Generic.Queue`1<T>.Count is the number of elements that are  
15 actually in the System.Collections.Generic.Queue`1<T>.

16  
17 The capacity is always greater than or equal to  
18 System.Collections.Generic.Queue`1<T>.Count. If  
19 System.Collections.Generic.Queue`1<T>.Count exceeds the capacity while adding  
20 elements, the capacity is increased by automatically reallocating the internal array  
21 before copying the old elements and adding the new elements.

22  
23 Retrieving the value of this property is an O(1) operation.

24

# 1 2 Queue<T>.System.Collections.ICollection.IsS 3 ynchronized Property

```
4 [ILAsm]  
5 .property instance bool System.Collections.ICollection.IsSynchronized  
6 [C#]  
7 bool System.Collections.ICollection.IsSynchronized { get; }
```

## 8 Summary

9 Gets a value indicating whether access to the System.Collections.ICollection is  
10 synchronized (thread safe).

## 11 Property Value

12 true if access to the System.Collections.ICollection is synchronized (thread safe);  
13 otherwise, false. In the default implementation of  
14 System.Collections.Generic.Queue<T>, this property always returns false.

## 15 Description

16 Default implementations of collections in System.Collections.Generic are not  
17 synchronized.

18  
19 Enumerating through a collection is intrinsically not a thread-safe procedure. To  
20 guarantee thread safety during enumeration, you can lock the collection during the  
21 entire enumeration. To allow the collection to be accessed by multiple threads for  
22 reading and writing, you must implement your own synchronization.

23  
24 System.Collections.ICollection.SyncRoot returns an object, which can be used to  
25 synchronize access to the System.Collections.ICollection. Synchronization is  
26 effective only if all threads lock this object before accessing the collection.

27  
28 Retrieving the value of this property is an O(1) operation.

29

# Queue<T>.System.Collections.ICollection.SyncRoot Property

```
[ILAsm]  
.property instance object System.Collections.ICollection.SyncRoot  
  
[C#]  
object System.Collections.ICollection.SyncRoot { get; }
```

## Summary

Gets an object that can be used to synchronize access to the System.Collections.ICollection.

## Property Value

An object that can be used to synchronize access to the System.Collections.ICollection. In the default implementation of System.Collections.Generic.Queue<T>, this property always returns the current instance.

## Description

Default implementations of collections in System.Collections.Generic are not synchronized.

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.

System.Collections.ICollection.SyncRoot returns an object, which can be used to synchronize access to the System.Collections.ICollection. Synchronization is effective only if all threads lock this object before accessing the collection. The following code shows the use of the System.Collections.ICollection.SyncRoot property for C#.

```
ICollection ic = ...;  
lock (ic.SyncRoot) {  
    // Access the collection.  
}
```

Retrieving the value of this property is an O(1) operation.