System.Collections.Generic.Queue<T> Class

```
[ILAsm]
.class public serializable beforefieldinit Queue`1<T> extends
System.Object implements System.Collections.Generic.IEnumerable`1<!0>,
System.Collections.ICollection, System.Collections.IEnumerable

[C#]
public class Queue<T>: System.Collections.Generic.IEnumerable<T>,
System.Collections.ICollection
```

Assembly Info:

- *Name:* System
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 4.0.0.0
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- **System.Collections.Generic.IEnumerable<T>**
- **System.Collections.ICollection**

Summary

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

Inherits From: System.Object

Library: BCL

Description

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

The capacity of a `System.Collections.Generic.Queue`1<T>` is the number of elements 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. The capacity can be decreased by calling `System.Collections.Generic.Queue`1<T>.TrimExcess`.

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

Queue<T>() Constructor

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

[C#]
public Queue ()
```

Summary

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

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(1) operation.

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
collection	The collection whose elements are copied to the new `System.Collections.Generic.Queue`1<T>`.

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
System.ArgumentNullException	*collection* 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
capacity	The initial number of elements that the `System.Collections.Generic.Queue`1<T>` 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
System.ArgumentOutOfRangeException	*capacity* is less than zero.

Queue<T>.Clear() Method

```
[ILAsm]
.method public hidebysig instance void Clear() cil managed

[C#]
public void Clear ()
```

Summary

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

Description

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

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

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

Queue<T>.Contains(T) Method

```
[ILAsm]
.method public hidebysig instance bool Contains(!0 item) cil managed

[C#]
public bool Contains (T item)
```

Summary

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

Parameters

Parameter	Description
item	The object to locate in the `System.Collections.Generic.Queue`1<T>`. The value can be `null` for reference types.

Return Value

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

Description

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

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

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

```
[ILAsm]
.method public hidebysig instance void CopyTo(!0[] array, int32
arrayIndex) cil managed

[C#]
public void CopyTo (T[] array, int arrayIndex)
```

Summary

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

Parameters

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

Description

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

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

Exceptions

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

	arrayIndex to the end of the destination *array*.

1

2

Queue<T>.Dequeue() Method

```
[ILAsm]
.method public hidebysig instance !0 Dequeue() cil managed

[C#]
public T Dequeue ()
```

Summary

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

Return Value

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

Description

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

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

This method is an O(1) operation.

Exceptions

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

Queue<T>.Enqueue(T) Method

```
[ILAsm]
.method public hidebysig instance void Enqueue(!0 item) cil managed

[C#]
public void Enqueue (T item)
```

Summary

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

Parameters

Parameter	Description
item	The object to add to the `System.Collections.Generic.Queue`1<T>`. The value can be `null` for reference types.

Description

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

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

Queue<T>.GetEnumerator() Method

```
[ILAsm]
.method public hidebysig instance valuetype
System.Collections.Generic.Queue`1/Enumerator<!0> GetEnumerator() cil
managed

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

Summary

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

Return Value

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

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.

Queue<T>.Peek() Method

```
[ILAsm]
.method public hidebysig instance !0 Peek() cil managed

[C#]
public T Peek ()
```

Summary

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

Return Value

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

Description

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

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

This method is an O(1) operation.

Exceptions

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

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

```
[ILAsm]
.method private hidebysig newslot virtual final instance class
System.Collections.Generic.IEnumerator`1<!0>
System.Collections.Generic.IEnumerable<T>.GetEnumerator() cil managed

[C#]
System.Collections.Generic.IEnumerator<T> IEnumerable<T>.GetEnumerator ()
```

Summary

Returns an enumerator that iterates through a collection.

Return Value

An `System.Collections.Generic.IEnumerator`1<T>` 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.

Queue<T>.System.Collections.ICollection.CopyTo(System.Array, System.Int32) Method

```
[ILAsm]
.method private hidebysig newslot virtual final instance void
System.Collections.ICollection.CopyTo(class System.Array array, int32
index) cil managed

[C#]
void ICollection.CopyTo (Array array, int index)
```

Summary

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

Parameters

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

Description

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

]

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`.

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

1

2

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.

Queue<T>.ToArray() Method

```
[ILAsm]
.method public hidebysig instance !0[] ToArray() cil managed

[C#]
public T[] ToArray ()
```

Summary

Copies the `System.Collections.Generic.Queue`1<T>` elements to a new array.

Return Value

A new array containing elements copied from the `System.Collections.Generic.Queue`1<T>`.

Description

The `System.Collections.Generic.Queue`1<T>` is not modified. The order of the elements in the new array is the same as the order of the elements from the beginning of the `System.Collections.Generic.Queue`1<T>` to its end.

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

Queue<T>.TrimExcess() Method

```
[ILAsm]
.method public hidebysig instance void TrimExcess() cil managed

[C#]
public void TrimExcess ()
```

Summary

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

Description

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

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

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

Queue<T>.Count Property

```
[ILAsm]
.property instance int32 Count

[C#]
public int Count { get; }
```

Summary

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

Property Value

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

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 store. `System.Collections.Generic.Queue`1<T>.Count` is the number of elements that are actually in the `System.Collections.Generic.Queue`1<T>`.

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

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

Queue<T>.System.Collections.ICollection.IsSynchronized Property

```
[ILAsm]
.property instance bool System.Collections.ICollection.IsSynchronized

[C#]
bool System.Collections.ICollection.IsSynchronized { get; }
```

Summary

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

Property Value

`true` if access to the `System.Collections.ICollection` is synchronized (thread safe); otherwise, `false`. In the default implementation of `System.Collections.Generic.Queue`1<T>`, this property always returns `false`.

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.

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

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`1<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.