

Queue<T>.Peek Method

Namespace: [System.Collections.Generic](#)

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

In this article

[Definition](#)

[Examples](#)

[Remarks](#)

[Applies to](#)

[See also](#)

Returns the object at the beginning of the [Queue<T>](#) without removing it.

C#

 Copy

```
public T Peek ();
```

Returns

T

The object at the beginning of the [Queue<T>](#).

Exceptions

[InvalidOperationException](#)

The [Queue<T>](#) is empty.

Examples

The following code example demonstrates several methods of the [Queue<T>](#) generic class, including the [Peek](#) method.

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

```
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 method is similar to the [Dequeue](#) method, but [Peek](#) does not modify the [Queue<T>](#).

If type `T` is a reference type, `null` can be added to the [Queue<T>](#) as a value.

This method is an O(1) operation.

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

See also

- [Enqueue\(T\)](#)
- [Dequeue\(\)](#)