

1 System.Collections.Generic.Stack<T> Class

```
2 [ILAsm]
3 .class public serializable beforefieldinit Stack`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 Stack<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 variable size last-in-first-out (LIFO) collection of instances of the same
20 arbitrary type.

21 Inherits From: System.Object

22
23 **Library:** BCL

25 Description

26 System.Collections.Generic.Stack`1<T> is implemented as an array.

27
28 The capacity of a System.Collections.Generic.Stack`1<T> is the number of elements
29 the System.Collections.Generic.Stack`1<T> can hold. As elements are added to a
30 System.Collections.Generic.Stack`1<T>, the capacity is automatically increased as
31 required by reallocating the internal array. The capacity can be decreased by calling
32 System.Collections.Generic.Stack`1<T>.TrimExcess.

33
34 If System.Collections.Generic.Stack`1<T>.Count is less than the capacity of the
35 stack, System.Collections.Generic.Stack`1<T>.Push is an O(1) operation. If the
36 capacity needs to be increased to accommodate the new element,
37 System.Collections.Generic.Stack`1<T>.Push becomes an O(*n*) operation, where *n*
38 is System.Collections.Generic.Stack`1<T>.Count.
39 System.Collections.Generic.Stack`1<T>.Pop is an O(1) operation.

40

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

3

1 Stack<T>() Constructor

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

7 Summary

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

10 Description

11 The capacity of a `System.Collections.Generic.Stack`1<T>` is the number of elements
12 that the `System.Collections.Generic.Stack`1<T>` can hold. As elements are added to
13 a `System.Collections.Generic.Stack`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.Stack`1<T>`.

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

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

24

Stack<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 Stack (System.Collections.Generic.IEnumerable<T> collection)
```

Summary

Initializes a new instance of the `System.Collections.Generic.Stack`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 to copy elements from.

Description

The capacity of a `System.Collections.Generic.Stack`1<T>` is the number of elements that the `System.Collections.Generic.Stack`1<T>` can hold. As elements are added to a `System.Collections.Generic.Stack`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.Stack`1<T>`.

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

The elements are copied onto the `System.Collections.Generic.Stack`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

1 Stack<T> (System.Int32) Constructor

```
2 [ILAsm]  
3 .method public hidebysig specialname rtspecialname instance void  
4 .ctor(int32 capacity) cil managed  
  
5 [C#]  
6 public Stack (int capacity)
```

7 Summary

8 Initializes a new instance of the `System.Collections.Generic.Stack<T>` class that is
9 empty and has the specified initial capacity or the default initial capacity, whichever is
10 greater.

11 Parameters

Parameter	Description
<i>capacity</i>	The initial number of elements that the <code>System.Collections.Generic.Stack<T></code> can contain.

12 13 Description

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

18
19 If the size of the collection can be estimated, specifying the initial capacity eliminates
20 the need to perform a number of resizing operations while adding elements to the
21 `System.Collections.Generic.Stack<T>`.

22
23 The capacity can be decreased by calling
24 `System.Collections.Generic.Stack<T>.TrimExcess`.

25
26 This constructor is an $O(n)$ operation, where n is *capacity*.

27 Exceptions

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

28

29

1 Stack<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.Stack`1<T>`.

8 Description

9 `System.Collections.Generic.Stack`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.Stack`1<T>`, call
14 `System.Collections.Generic.Stack`1<T>.TrimExcess`. Trimming an empty
15 `System.Collections.Generic.Stack`1<T>` sets the capacity of the
16 `System.Collections.Generic.Stack`1<T>` to the default capacity.

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

20

1 Stack<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.Stack`1<T>`.

8 Parameters

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

9 Return Value

11 true if *item* is found in the `System.Collections.Generic.Stack`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 list.

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

20

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

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

7 Summary

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

10 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.Stack<T></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.

11 12 Description

13 The elements are copied onto the array in last-in-first-out (LIFO) order, similar to the
14 order of the elements returned by a succession of calls to
15 `System.Collections.Generic.Stack<T>.Pop`.

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

19 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>arrayIndex</i> is less than zero.
System.ArgumentException	The number of elements in the source <code>System.Collections.Generic.Stack<T></code> is greater than the available space from <i>arrayIndex</i> to the end of the destination <i>array</i> .

1

2

1 Stack<T>.GetEnumerator() Method

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

8 Summary

9 Returns an enumerator for the `System.Collections.Generic.Stack`1<T>`.

10 Return Value

11 An `System.Collections.Generic.Stack`1<T>.Enumerator` for the
12 `System.Collections.Generic.Stack`1<T>`.

13 Description

14 Usage

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

17
18
19 Default implementations of collections in `System.Collections.Generic` are not
20 synchronized.

21
22 This method is an O(1) operation.

23

1 Stack<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 top of the `System.Collections.Generic.Stack<T>` without
8 removing it.

9 Return Value

10 The object at the top of the `System.Collections.Generic.Stack<T>`.

11 Description

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

15
16 If type *T* is a reference type, null can be pushed onto the
17 `System.Collections.Generic.Stack<T>` as a placeholder, if needed.

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

20 Exceptions

Exception	Condition
<code>System.InvalidOperationException</code>	The <code>System.Collections.Generic.Stack<T></code> is empty.

21

22

1 Stack<T>.Pop() Method

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

6 Summary

7 Removes and returns the object at the top of the
8 System.Collections.Generic.Stack<T>.

9 Return Value

10 The object removed from the top of the System.Collections.Generic.Stack<T>.

11 Description

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

15
16 If type *T* is a reference type, null can be pushed onto the
17 System.Collections.Generic.Stack<T> as a placeholder, if needed.

18
19 System.Collections.Generic.Stack<T> is implemented as an array. This method is
20 an O(1) operation.

21 Exceptions

Exception	Condition
System.InvalidOperationException	The System.Collections.Generic.Stack<T> is empty.

22

23

1 Stack<T>.Push(T) Method

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

6 Summary

7 Inserts an object at the top of the `System.Collections.Generic.Stack<T>`.

8 Parameters

Parameter	Description
<i>item</i>	The object to push onto the <code>System.Collections.Generic.Stack<T></code> . The value can be null for reference types.

9

10 Description

11 `System.Collections.Generic.Stack<T>` is implemented as an array.

12

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

17

18 If type *T* is a reference type, null can be pushed onto the
19 `System.Collections.Generic.Stack<T>` as a placeholder, if needed. It occupies a
20 slot in the stack and is treated like any object.

21

22 If `System.Collections.Generic.Stack<T>.Count` is less than the capacity of the
23 stack, `System.Collections.Generic.Stack<T>.Push` is an $O(1)$ operation. If the
24 capacity needs to be increased to accommodate the new element,
25 `System.Collections.Generic.Stack<T>.Push` becomes an $O(n)$ operation, where *n*
26 is `System.Collections.Generic.Stack<T>.Count`.

27

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

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

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

Summary

Returns an enumerator that iterates through the 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.

1
2 **Stack<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 arrayIndex) cil managed  
  
8 [C#]  
9 void ICollection.CopyTo (Array array, int arrayIndex)
```

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>arrayIndex</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.Stack`1<T>.Count`.

25 **Exceptions**

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

System.ArgumentOutOfRangeException	<i>arrayIndex</i> is less than zero.
System.ArgumentException	<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>arrayIndex</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

Stack<T>.System.Collections.IEnumerable.Get 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.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 Stack<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.Stack<T>` to a new array.

8 Return Value

9 A new array containing copies of the elements of the
10 `System.Collections.Generic.Stack<T>`.

11 Description

12 The elements are copied onto the array in last-in-first-out (LIFO) order, similar to the
13 order of the elements returned by a succession of calls to
14 `System.Collections.Generic.Stack<T>.Pop`.

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

18

1 Stack<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.Stack<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.Stack<T> can be considerable, however, so the
14 System.Collections.Generic.Stack<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
18 This method is an $O(n)$ operation, where n is
19 System.Collections.Generic.Stack<T>.Count.

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

26

1 Stack<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.Stack<T>.

9 Property Value

10 The number of elements contained in the System.Collections.Generic.Stack<T>.

11 Description

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

16
17 The capacity is always greater than or equal to
18 System.Collections.Generic.Stack<T>.Count. If
19 System.Collections.Generic.Stack<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

Stack<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.Stack<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. In the rare case where enumerations contend with write accesses, you must 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 that 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.

29

Stack<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.Stack<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 that 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.