

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

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

11 Assembly Info:

- 12 • *Name:* mscorlib
- 13 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- 14 • *Version:* 2.0.x.x
- 15 • *Attributes:*
 - 16 ○ CLSCompliantAttribute(true)

17 Implements:

- 18 • **System.Collections.ICollection**
- 19 • **System.Collections.IEnumerable**
- 20 • **System.Collections.IList<T>**
- 21 • **System.Collections.Generic.ICollection<T>**
- 22 • **System.Collections.Generic.IEnumerable<T>**
- 23 • **System.Collections.Generic.IList<T>**

24 Summary

25 Implements the `System.Collections.Generic.IList<T>` interface. The size of a List is
26 dynamically increased as required. A List is not guaranteed to be sorted. It is the
27 programmer's responsibility to sort the List prior to performing operations (such as
28 `BinarySearch`) that require a List to be sorted. Indexing operations are required to
29 perform in constant access time; that is, $O(1)$.

30 Inherits From: System.Object

31

32 **Library:** BCL

33

34 **Thread Safety:** Static members of this type are thread-safe. Any instance members are
35 not guaranteed to be thread safe. A list can support multiple readers concurrently, as long
36 as the collection is not modified. Even so, enumerating through a collection is intrinsically
37 not a thread safe procedure. [Note: To guarantee thread safety during enumeration, you
38 can lock the collection during the entire enumeration. To allow the collection to be accessed
39 by multiple threads for reading and writing, you must implement your own synchronization.]

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
26
27
28
29
30
31
32
33
34

Description

Some methods, such as `Contains`, `IndexOf`, `LastIndexOf`, and `Remove`, use an equality comparer for the list elements. The default equality comparer for type `T` is determined as follows: If type `T` implements `System.IEquatable<T>` then the default equality comparer is `System.IEquatable<T>.Equals(T)`; otherwise the default equality comparer is `System.Object.Equals(Object)`.

Some methods, such as `BinarySearch` and `Sort`, use a comparer for the list elements. Some overloads of these methods take an explicit comparer as argument, while others use a default comparer. The default comparer for type `T` is determined as follows: If type `T` implements `System.IComparable<T>` then the default comparer is `System.IComparable<T>.CompareTo(T)`; otherwise, if type `T` implements `System.IComparable` then the default comparer is `System.IComparable.CompareTo(Object)`. If type `T` implements neither `System.IComparable<T>` nor `System.IComparable` then there is no default comparer; in this case a comparer or comparison delegate must be given explicitly.

The capacity of a `System.Collections.Generic.List<T>` is the number of elements the `System.Collections.Generic.List<T>` can hold. As elements are added to a `System.Collections.Generic.List<T>`, the capacity is automatically increased as required.. The capacity can be decreased by calling `System.Collections.Generic.List<T>.TrimToSize` or by setting the `System.Collections.Generic.List<T>.Capacity` property explicitly.

Indexes in this collection are zero-based.

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

This type contains a member that is a nested type, called `Enumerator`. Although `Enumerator` is a member of this type, `Enumerator` is not described here; instead, it is described in its own entry, `List<T>.Enumerator`.

1 List<T>() Constructor

```
2 [ILAsm]  
3 public rtspecialname specialname instance void .ctor()  
4 [C#]  
5 public List()
```

6 Summary

7 Initializes a new list that is empty and has the default initial capacity.

8 Description

9 [Note: If the size of the collection can be estimated, you can specify the initial capacity
10 in a constructor overload that accepts a capacity parameter to eliminate the need to
11 perform a number of resizing operations while adding elements to the list.]
12
13

14

1
2 **List<T> (System.Collections.Generic.IEnumerable<T>) Constructor**
3

```
4 [ILAsm]  
5 public rtspecialname specialname instance void .ctor(class  
6 System.Collections.Generic.IEnumerable`1<!0> collection)  
  
7 [C#]  
8 public List(IEnumerable<T> collection)
```

9 **Summary**

10 Initializes a new list with elements copied from the specified collection, ensuring that the
11 list has sufficient capacity to accommodate the number of elements copied.

12 **Parameters**

Parameter	Description
<i>collection</i>	The collection from which to copy the elements.

13
14 **Description**

15 [Note: If the size of the collection can be estimated, you can specify the initial capacity
16 in a constructor overload that accepts a capacity parameter to eliminate the need to
17 perform a number of resizing operations while adding elements to the list.]
18

19
20
21 The elements are copied onto the list in the same order in which they are read by the
22 System.Collections.Generic.IEnumerator<T> from collection.

23 **Exceptions**

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

24
25

1 List<T> (System.Int32) Constructor

```
2 [ILAsm]  
3 public rtspecialname specialname instance void .ctor(int32 capacity)  
4 [C#]  
5 public List(int capacity)
```

6 Summary

7 Initializes a new list that is empty and has the specified initial capacity.

8 Parameters

Parameter	Description
<i>capacity</i>	The maximum number of elements that the List can contain without reallocating memory.

9

10 Exceptions

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

11

12

1 List<T>.Add(T) Method

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

6 Summary

7 Adds an item to the end of the list.

8 Parameters

Parameter	Description
<i>item</i>	The item to add to the end of the list. (<i>item</i> can be null if T is a reference type.)

9

10 Description

11 System.Collections.Generic.List<T> accepts null as a valid value for reference
12 types and allows duplicate elements.

13

14 If System.Collections.Generic.List<T>.Count already equals
15 System.Collections.Generic.List<T>.Capacity, the capacity of the list is increased.

16

17 If System.Collections.Generic.List<T>.Count is less than
18 System.Collections.Generic.List<T>.Capacity, this method is an O(1) operation. If
19 the capacity needs to be increased to accommodate the new element, this method
20 becomes an O(n) operation, where n is System.Collections.Generic.List<T>.Count.

21

1 2 List<T>.AddRange(System.Collections.Generic 3 c.IEnumerable<T>) Method

```
4 [ILAsm]  
5 .method public hidebysig void AddRange(class  
6 System.Collections.Generic.IEnumerable`1<!0> collection)  
  
7 [C#]  
8 public void AddRange(IEnumerable<T> collection)
```

9 Summary

10 Adds the elements of the specified collection to the end of the list.

11 Parameters

Parameter	Description
<i>collection</i>	The collection whose elements are added to the end of the list.

12 13 Description

14 System.Collections.Generic.List<T> accepts null as a valid value for reference
15 types and allows duplicate elements.

16
17 The order of the elements in the collection is preserved in the
18 System.Collections.Generic.List<T>.

19
20 If the new System.Collections.Generic.List<T>.Count (the current
21 System.Collections.Generic.List<T>.Count plus the size of the collection) will be
22 greater than System.Collections.Generic.List<T>.Capacity, the capacity of the list
23 is increased.

24
25 If the list can accommodate the new elements without increasing
26 System.Collections.Generic.List<T>.Capacity, this method is an O(n) operation,
27 where n is the number of elements to be added. If the capacity needs to be increased to
28 accommodate the new elements, this method becomes an O(n + m) operation, where n
29 is the number of elements to be added and m is
30 System.Collections.Generic.List<T>.Count.

31 Exceptions

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

1

2

1 List<T>.AsReadOnly() Method

```
2 [ILAsm]  
3 .method public hidebysig class System.Collections.Generic.IList`1<!0>  
4 AsReadOnly()  
5 [C#]  
6 public IList<T> AsReadOnly()
```

7 Summary

8 Returns a read-only wrapper to the current List.

9 Return Value

10 A read-only wrapper for the current List.

11 Description

12 To prevent any modifications to a list, expose it only through this wrapper.

13
14 A collection that is read-only is simply a collection with a wrapper that prevents
15 modifying the collection; therefore, if changes are made to the underlying collection, the
16 read-only collection reflects those changes.

17

List<T>.BinarySearch(T) Method

```
[ILAsm]  
.method public hidebysig int32 BinarySearch(!0 item)  
  
[C#]  
public int BinarySearch(T item)
```

Summary

Searches the entire sorted list for an element using the default comparer, and returns the zero-based index of the element.

Parameters

Parameter	Description
<i>item</i>	The element for which to search. (<i>item</i> can be null if T is a reference type.)

Return Value

The zero-based index of *item* in the sorted list, if *item* is found; otherwise, a negative number, which is the bitwise complement of the index of the next element that is larger than *item* or, if there is no larger element, the bitwise complement of `System.Collections.Generic.List<T>.Count`.

Description

This method uses the default comparer for type T to determine the order of list elements. If there is no default comparer, then the method throws `System.InvalidOperationException`. The default comparer for a given element type T is defined in the Description section of this (class `List<T>`) specification.

The list must already be sorted according to the comparer implementation; otherwise, the result is incorrect.

Comparing `null` with any reference type is allowed and does not generate an exception when using `System.IComparable<T>`. When sorting, `null` is considered to be less than any other object.

If the list contains more than one element with the same value, the method returns only one of the occurrences, and it might return any one of the occurrences, not necessarily the first one.

If the list does not contain the specified value, the method returns a negative integer. You can apply the bitwise complement operation (`~`) to this negative integer to get the index of the first element that is larger than the search value. When inserting the value into the list, this index should be used as the insertion point to maintain the sort order.

1
2 This method is an $O(\log n)$ operation, where n is the number of elements in the list.

3 **Exceptions**

Exception	Condition
System.InvalidOperationException	The default comparer cannot find a <code>System.IComparable<T></code> or <code>System.IComparable</code> implementation for type <code>T</code> .

4
5

1 List<T>.BinarySearch(T, 2 System.Collections.Generic.IComparer<T>) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig int32 BinarySearch(!0 item, class  
6 System.Collections.Generic.IComparer`1<!0> comparer)  
7  
8 [C#]  
9 public int BinarySearch(T item, IComparer<T> comparer)
```

9 Summary

10 Searches the entire sorted list for an element using the specified comparer and returns
11 the zero-based index of the element.

12 Parameters

Parameter	Description
<i>item</i>	The element for which to search. (<i>item</i> can be null if T is a reference type.)
<i>comparer</i>	The System.Collections.Generic.IComparer<T> implementation to use when comparing elements. -or- null to use the default comparer.

13

14 Return Value

15 The zero-based index of *item* in the sorted list, if *item* is found; otherwise, a negative
16 number, which is the bitwise complement of the index of the next element that is larger
17 than *item* or, if there is no larger element, the bitwise complement of
18 System.Collections.Generic.List<T>.Count.

19 Description

20 If the given comparer is non-null, it is used to determine the order of list elements. If
21 the given comparer is null, the default comparer for type T is used; if there is no
22 default comparer, then the method throws System.InvalidOperationException. The
23 default comparer for a given element type T is defined in the Description section of this
24 (class List<T>) specification.

25

26 The comparer customizes how the elements are compared. For example, if T is
27 System.String, you can use a System.Collections.CaseInsensitiveComparer

1 instance as the comparer to perform case-insensitive string searches.

2

3 The list must already be sorted according to the comparer implementation; otherwise,
4 the result is incorrect.

5

6 Comparing `null` with any reference type is allowed and does not generate an exception
7 when using `System.IComparable<T>`. When sorting, `null` is considered to be less than
8 any other object.

9

10 If the `System.Collections.Generic.List<T>` contains more than one element with the
11 same value, the method returns only one of the occurrences, and it might return any
12 one of the occurrences, not necessarily the first one.

13

14 If the list does not contain the specified value, the method returns a negative integer.
15 You can apply the bitwise complement operation (`~`) to this negative integer to get the
16 index of the first element that is larger than the search value. When inserting the value
17 into the list, this index should be used as the insertion point to maintain the sort order.

18

19 This method is an $O(\log n)$ operation, where n is the number of elements in the list.

20 Exceptions

Exception	Condition
System.InvalidOperationException	<i>comparer</i> is null, and the default comparer cannot find a <code>System.IComparable<T></code> or <code>System.IComparable</code> implementation for type <code>T</code> .

21

22

1 **List<T>.BinarySearch(System.Int32,**
2 **System.Int32, T,**
3 **System.Collections.Generic.IComparer<T>)**
4 **Method**

```
5 [ILAsm]  
6 .method public hidebysig int32 BinarySearch(int32 index, int32 count, !0  
7 item, class System.Collections.Generic.IComparer`1<!0> comparer)  
  
8 [C#]  
9 public int BinarySearch(int index, int count, T item, IComparer<T>  
10 comparer)
```

11 **Summary**

12 Searches a range of elements in the sorted list for an element using the specified
13 comparer and returns the zero-based index of the element.

14 **Parameters**

Parameter	Description
<i>index</i>	The zero-based starting index of the range to search.
<i>count</i>	The length of the range to search.
<i>item</i>	The element for which to search. (<i>item</i> can be null if T is a reference type.)
<i>comparer</i>	The System.Collections.Generic.IComparer<T> implementation to use when comparing elements. -or- null to use the default comparer.

15
16 **Return Value**

17 The zero-based index of *item* in the sorted list, if *item* is found; otherwise, a negative
18 number, which is the bitwise complement of the index of the next element that is larger
19 than *item* or, if there is no larger element, the bitwise complement of *index* + *count*.

20 **Description**

1 If the given comparer is non-null, it is used to determine the order of list elements. If
2 the given comparer is null, the default comparer for type T is used; if there is no
3 default comparer, then the method throws `System.InvalidOperationException`. The
4 default comparer for a given element type T is defined in the Description section of this
5 (class `List<T>`) specification.

6
7 The comparer customizes how the elements are compared. For example, if T is
8 `System.String`, you can use a `System.Collections.CaseInsensitiveComparer`
9 instance as the comparer to perform case-insensitive string searches.

10
11 The list must already be sorted according to the comparer implementation; otherwise,
12 the result is incorrect.

13
14 Comparing null with any reference type is allowed and does not generate an exception
15 when using `System.IComparable<T>`. When sorting, null is considered to be less than
16 any other object.

17
18 If the `System.Collections.Generic.List<T>` contains more than one element with the
19 same value, the method returns only one of the occurrences, and it might return any
20 one of the occurrences, not necessarily the first one.

21
22 If the list does not contain the specified value, the method returns a negative integer.
23 You can apply the bitwise complement operation (`~`) to this negative integer to get the
24 index of the first element that is larger than the search value. When inserting the value
25 into the list, this index should be used as the insertion point to maintain the sort order.

26
27 This method is an $O(\log n)$ operation, where n is the number of elements in the range.

28 Exceptions

Exception	Condition
System.ArgumentException	<i>index</i> + <i>count</i> is greater than <code>System.Collections.Generic.List<T>.Count</code> .
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>count</i> is less than zero.
System.InvalidOperationException	<i>comparer</i> is null, and the default comparer cannot find a <code>System.IComparable<T></code> or <code>System.IComparable</code> implementation for type T.

29

30

1 List<T>.Clear() Method

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

6 Summary

7 Removes all elements from the list.

8 Description

9 System.Collections.Generic ICollection<T>.Count gets set to zero, and references
10 to other objects from elements of the collection are also released. The capacity remains
11 unchanged.

12
13 [*Note:* To reset the capacity, call System.Collections.Generic.List<T>.TrimToSize
14 or set the System.Collections.Generic.List<T>.Capacity property directly.

15
16]

17
18

19

1 List<T>.Contains(T) Method

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

6 Summary

7 Determines whether the list contains a specific value.

8 Parameters

Parameter	Description
<i>item</i>	The object to locate in the current collection. (<i>item</i> can be null if T is a reference type.)

9 Return Value

10 true, if *item* is found in the list; otherwise, false.

12 Description

13 This method uses the default equality comparer for type T to determine equality of list
14 elements. The default equality comparer for element type T is defined in the Description
15 section of this (class List<T>) specification.

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

19

1 List<T>.ConvertAll(System.Converter<T,U>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig System.Collections.Generic.List`1<!1>  
5 ConvertAll<U>(class System.Converter`2<!0,!1> converter)  
  
6 [C#]  
7 public List<U> ConvertAll<U>(Converter<T,U> converter)
```

8 Summary

9 Converts the current List (of type T) to a List of type U.

10 Parameters

Parameter	Description
<i>converter</i>	A converter delegate that converts each element from one type to another type.

11 Return Value

12 A List of the target type containing the converted elements from the current List.

14 Description

15 The converter is a delegate that converts an object to the target type. The elements of
16 the current List are individually passed to the converter delegate, and the converted
17 elements are saved in the new List.

18 The current List remains unchanged.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>converter</i> is null.

21
22

1 List<T>.CopyTo(T[]) Method

```
2 [ILAsm]  
3 .method public hidebysig void CopyTo(!0[] array)  
4 [C#]  
5 public void CopyTo(T[] array)
```

6 Summary

7 Copies the entire list to an array.

8 Parameters

Parameter	Description
<i>array</i>	A one-dimensional, zero-based array that is the destination of the elements copied from the list.

9

10 Description

11 The elements are copied onto the array (using `System.Array.Copy`) in the same order
12 in which the enumerator iterates through the list.

13 Exceptions

Exception	Condition
System.ArgumentException	<i>array</i> is multidimensional. -or- <i>array</i> does not have zero-based indexing. -or- The number of elements in the list is greater than the number of elements that the destination <i>array</i> can contain. -or- Type T is not assignable to the element type of the destination array.

System.ArgumentNullException	<i>array</i> is null.
-------------------------------------	-----------------------

1

2

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

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

6 Summary

7 Copies the elements of the list to an array, starting at a particular index.

8 Parameters

Parameter	Description
<i>array</i>	A one-dimensional, zero-based array that is the destination of the elements copied from the list.
<i>arrayIndex</i>	The zero-based index in <i>array</i> at which copying begins.

9

10 Description

11 The elements are copied onto the array (using *System.Array.Copy*) in the same order
12 in which the enumerator iterates through the list.

13 Exceptions

Exception	Condition
System.ArgumentException	<i>array</i> is multidimensional. -or- <i>array</i> does not have zero-based indexing. -or- The sum of <i>arrayIndex</i> and number of elements in the list is greater than the length of the destination array. -or- Type T is not assignable to the element type of

	the destination array.
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>arrayIndex</i> is less than zero.

1

2

1 List<T>.CopyTo(System.Int32, T[], 2 System.Int32, System.Int32) Method

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

8 Summary

9 Copies a range of elements of the list to an array, starting at a particular index in the
10 target array.

11 Parameters

Parameter	Description
<i>index</i>	The zero-based index in the source list at which copying begins.
<i>array</i>	A one-dimensional, zero-based array that is the destination of the elements copied from the list.
<i>arrayIndex</i>	The zero-based index in <i>array</i> at which copying begins.
<i>count</i>	The number of elements to copy.

12

13 Description

14 The elements are copied onto the array (using `System.Array.Copy`) in the same order
15 in which the enumerator iterates through the list.

16 Exceptions

Exception	Condition
System.ArgumentException	<i>array</i> is multidimensional. -or- <i>index</i> is equal to or greater than the <code>System.Collections.Generic.List<T>.Count</code> of the source list.

	<p>-or-</p> <p><i>arrayIndex</i> is equal to or greater than the length of <i>array</i>.</p> <p>-or-</p> <p>The number of elements from <i>index</i> to the end of the source list is greater than the available space from <i>arrayIndex</i> to the end of the destination <i>array</i>.</p> <p>-or-</p> <p>Type T is not assignable to the element type of the destination array.</p>
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<p><i>index</i> is less than zero.</p> <p>-or-</p> <p><i>array</i> does not have zero-based indexing.</p> <p>-or-</p> <p><i>arrayIndex</i> is less than zero.</p> <p>-or-</p> <p><i>count</i> is less than zero.</p>

1

2

1 List<T>.Exists(System.Predicate<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig bool Exists(class System.Predicate`1<!0> match)  
  
5 [C#]  
6 public bool Exists(Predicate<T> match)
```

7 Summary

8 Determines whether the List contains elements that match the conditions defined by the
9 specified predicate.

10 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the elements to search for.

11 Return Value

12 `true` if the List contains one or more elements that match the conditions defined by the
13 specified predicate; otherwise, `false`.

15 Description

16 The predicate is a delegate that returns `true` if the object passed to it matches the
17 conditions defined in the delegate. The elements of the current List are individually
18 passed to the predicate delegate, and processing is stopped when a match is found.

19 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

20
21

1 List<T>.Find(System.Predicate<T>) Method

```
2 [ILAsm]  
3 .method public hidebysig !0 Find(class System.Predicate`1<!0> match)  
4 [C#]  
5 public T Find(Predicate<T> match)
```

6 Summary

7 Searches for an element that matches the conditions defined by the specified predicate,
8 and returns the first occurrence within the entire List.

9 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the element to search for.

10

11 Return Value

12 The first element that matches the conditions defined by the specified predicate, if
13 found; otherwise, the default value for type T.

14 Description

15 The predicate is a delegate that returns `true` if the object passed to it matches the
16 conditions defined in the delegate. The elements of the current List are individually
17 passed to the predicate delegate, moving forward in the List, starting with the first
18 element and ending with the last element. Processing is stopped when a match is found.

19 Exceptions

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

20

21

1 List<T>.FindAll(System.Predicate<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig System.Collections.Generic.List`1<!0>  
5 FindAll(class System.Predicate<!0> match)  
  
6 [C#]  
7 public List<T> FindAll(Predicate<T> match)
```

8 Summary

9 Retrieves all the elements that match the conditions defined by the specified predicate.

10 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the elements to search for.

11 Return Value

12 A List containing all the elements that match the conditions defined by the specified
13 predicate, if found; otherwise, an empty List.

15 Description

16 The predicate is a delegate that returns `true` if the object passed to it matches the
17 conditions defined in the delegate. The elements of the current List are individually
18 passed to the Predicate delegate, and the elements that match the conditions are saved
19 in the returned List.

20 Exceptions

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

21
22

1 List<T>.FindIndex(System.Predicate<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig int32 FindIndex(class System.Predicate`1<!0>  
5 match)  
6 [C#]  
7 public int FindIndex(Predicate<T> match)
```

8 Summary

9 Searches for an element that matches the conditions defined by the specified predicate,
10 and returns the zero-based index of the first occurrence within the List.

11 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the element to search for.

12 Return Value

14 The zero-based index of the first occurrence of an element that matches the conditions
15 defined by *match*, if found; otherwise, -1.

16 Description

17 The List is searched forward starting at the first element and ending at the last element.
18
19 The predicate is a delegate that returns `true` if the object passed to it matches the
20 conditions defined in the delegate. The elements of the current List are individually
21 passed to the predicate delegate.

22 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

23
24

1 List<T>.FindIndex(System.Int32, 2 System.Predicate<T>) Method

```
3 [ILAsm]  
4 .method public hidebysig int32 FindIndex(int32 index, class  
5 System.Predicate`1<!0> match)  
  
6 [C#]  
7 public int FindIndex(int index, Predicate<T> match)
```

8 Summary

9 Searches for an element that matches the conditions defined by the specified predicate,
10 and returns the zero-based index of the first occurrence within the range of elements in
11 the List that extends from the specified index to the last element.

12 Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the search.
<i>match</i>	The predicate delegate that specifies the element to search for.

13 14 Return Value

15 The zero-based index of the first occurrence of an element that matches the conditions
16 defined by *match*, if found; otherwise, -1.

17 Description

18 The List is searched forward starting at *index* and ending at the last element.

19
20 The predicate is a delegate that returns `true` if the object passed to it matches the
21 conditions defined in the delegate. The elements of the current List are individually
22 passed to the predicate delegate.

23 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.
System.ArgumentOutOfRangeException	<i>index</i> is less than 0 or greater than or equal to <code>System.Collections.Generic.List<T>.Count</code> .

1

2

1 List<T>.FindIndex(System.Int32, 2 System.Int32, System.Predicate<T>) Method

```
3 [ILAsm]  
4 .method public hidebysig int32 FindIndex(int32 index, int32 count, class  
5 System.Predicate`1<!0> match)  
  
6 [C#]  
7 public int FindIndex(int index, int count, Predicate<T> match)
```

8 Summary

9 Searches for an element that matches the conditions defined by the specified predicate,
10 and returns the zero-based index of the first occurrence within the range of elements in
11 the List that starts at the specified index and contains the specified number of elements.

12 Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the search.
<i>count</i>	The number of elements to search.
<i>match</i>	The predicate delegate that specifies the element to search for.

13

14 Return Value

15 The zero-based index of the first occurrence of an element that matches the conditions
16 defined by *match*, if found; otherwise, -1.

17 Description

18 The List is searched forward starting at *index* and ending after *count* elements.

19

20 The predicate is a delegate that returns `true` if the object passed to it matches the
21 conditions defined in the delegate. The elements of the current List are individually
22 passed to the predicate delegate.

23 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

System.ArgumentOutOfRangeException

index is less than 0.

-or-

count is less than 0.

-or-

index + *count* is greater than

`System.Collections.Generic.List<T>.Count`.

1

2

1 List<T>.FindLast(System.Predicate<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig !0 FindLast(class System.Predicate`1<!0> match)  
5 [C#]  
6 public T FindLast(Predicate<T> match)
```

7 Summary

8 Searches for an element that matches the conditions defined by the specified predicate,
9 and returns the last occurrence within the entire List.

10 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the element to search for.

11 Return Value

13 The last element that matches the conditions defined by the specified predicate, if
14 found; otherwise, the default value for type T.

15 Description

16 The predicate is a delegate that returns `true` if the object passed to it matches the
17 conditions defined in the delegate. The elements of the current List are individually
18 passed to the predicate delegate, moving backward in the List, starting with the last
19 element and ending with the first element. Processing is stopped when a match is found.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

21
22

1
2 **List<T>.FindLastIndex(System.Predicate<T>**
3 **) Method**

```
4 [ILAsm]  
5 .method public hidebysig int32 FindLastIndex(class System.Predicate`1<!0>  
6 match)  
7 [C#]  
8 public int FindLastIndex(Predicate<T> match)
```

9 **Summary**

10 Searches for an element that matches the conditions defined by the specified predicate,
11 and returns the zero-based index of the last occurrence within the List.

12 **Parameters**

Parameter	Description
<i>match</i>	The predicate delegate that specifies the element to search for.

13
14 **Return Value**

15 The zero-based index of the last occurrence of an element that matches the conditions
16 defined by *match*, if found; otherwise, -1.

17 **Description**

18 The List is searched backward starting at the last element and ending at the first
19 element.

20
21 The predicate is a delegate that returns `true` if the object passed to it matches the
22 conditions defined in the delegate. The elements of the current List are individually
23 passed to the predicate delegate.

24 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

25
26

1 List<T>.FindLastIndex(System.Int32, 2 System.Predicate<T>) Method

```
3 [ILAsm]  
4 .method public hidebysig int32 FindLastIndex(int32 index, class  
5 System.Predicate`1<!0> match)  
  
6 [C#]  
7 public int FindLastIndex(int index, Predicate<T> match)
```

8 Summary

9 Searches for an element that matches the conditions defined by the specified predicate,
10 and returns the zero-based index of the last occurrence within the range of elements in
11 the List that extends from the specified index to the first element.

12 Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the backward search.
<i>match</i>	The predicate delegate that specifies the element to search for.

13

14 Return Value

15 The zero-based index of the last occurrence of an element that matches the conditions
16 defined by *match*, if found; otherwise, -1.

17 Description

18 The List is searched backward starting at *index* and ending at the first element.

19

20 The predicate is a delegate that returns `true` if the object passed to it matches the
21 conditions defined in the delegate. The elements of the current List are individually
22 passed to the predicate delegate.

23 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.
System.ArgumentOutOfRangeException	<i>index</i> is less than 0 or greater than or equal to <code>System.Collections.Generic.List<T>.Count</code> .

1

2

List<T>.FindLastIndex(System.Int32, System.Int32, System.Predicate<T>) Method

```
[ILAsm]
.method public hidebysig int32 FindLastIndex(int32 index, int32 count,
class System.Predicate`1<!0> match)

[C#]
public int FindLastIndex(int index, int count, Predicate<T> match)
```

Summary

Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the range of elements in the List that starts at the specified index and contains the specified number of elements going backwards.

Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the search.
<i>count</i>	The number of elements to search.
<i>match</i>	The predicate delegate that specifies the element to search for.

Return Value

The zero-based index of the last occurrence of an element that matches the conditions defined by *match*, if found; otherwise, -1.

Description

The List is searched backward starting at *index* and ending after *count* elements.

The predicate is a delegate that returns `true` if the object passed to it matches the conditions defined in the delegate. The elements of the current List are individually passed to the predicate delegate.

Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

System.ArgumentOutOfRangeException

index is less than zero, or greater than or equal to

`System.Collections.Generic.List<T>.Count`.

-or-

count is less than 0.

-or-

count is greater than *index* + 1.

1

2

1 List<T>.ForEach(System.Action<T>) Method

```
2 [ILAsm]  
3 .method public hidebysig void ForEach(class System.Action`1<!0> action)  
4 [C#]  
5 public void ForEach(Action<T> action)
```

6 Summary

7 Performs the specified action on each element of the List.

8 Parameters

Parameter	Description
<i>action</i>	The action delegate to perform on each element of the List.

9 10 Description

11 The action is a delegate that performs an action on the object passed to it. The elements
12 of the current List are individually passed to the action delegate, sequentially, in index
13 order, and on the same thread as that used to call ForEach. Execution stops if the
14 action throws an exception.

15 Exceptions

Exception	Condition
System.ArgumentNullException	<i>action</i> is null.

16
17

1 List<T>.GetEnumerator() Method

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

7 Summary

8 Returns an enumerator, in index order, that can be used to iterate over the list.

9 Return Value

10 An enumerator for the list.

11 Usage

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

14

15

1 List<T>.GetRange(System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig class System.Collections.Generic.List`1<!0>  
5 GetRange(int32 index, int32 count)  
  
6 [C#]  
7 public List<T> GetRange(int index, int count)
```

8 Summary

9 Creates a shallow copy of a range of elements in the current List.

10 Parameters

Parameter	Description
<i>index</i>	The zero-based index at which the range starts.
<i>count</i>	The number of elements in the range.

11

12 Return Value

13 A shallow copy of the given range of elements in the list.

14 Description

15 A shallow copy of a collection, or a subset of that collection, copies only the elements of
16 the collection, whether they are reference types or value types, but it does not copy the
17 objects that the references refer to. The references in the new collection point to the
18 same objects as do the references in the original collection. (In contrast, a deep copy of
19 a collection copies the elements and everything directly or indirectly referenced by those
20 elements.)

21 Exceptions

Exception	Condition
System.ArgumentException	<i>index</i> + <i>count</i> is greater than System.Collections.Generic.List<T>.Count.
System.ArgumentOutOfRangeException	<i>index</i> is less than 0. -or-

	<i>count</i> is less than 0.
--	------------------------------

1

2

1 List<T>.IndexOf(T) Method

```
2 [ILAsm]  
3 .method public hidebysig int32 IndexOf(!0 value)  
4 [C#]  
5 public int IndexOf(T value)
```

6 Summary

7 Searches for the specified object and returns the zero-based index of the first
8 occurrence within the entire list.

9 Parameters

Parameter	Description
<i>value</i>	The T to locate in the current list. (The value can be null if T is a reference type.)

10

11 Return Value

12 The zero-based index of the first occurrence of *item* within the List, if found; otherwise,
13 -1.

14 Description

15 The list is searched forward starting at the first element and ending at the last element.

16

17 This method uses the default equality comparer for type T to determine equality of list
18 elements. The default equality comparer for element type T is defined in the Description
19 section of this (class List<T>) specification.

20

21 This method performs a linear search; therefore, the average number of comparisons is
22 proportional to System.Collections.Generic.List<T>.Count. That is, this method is
23 an O(n) operation, where n is System.Collections.Generic.List<T>.Count.

24

1 List<T>.IndexOf(T, System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig int32 IndexOf(!0 value, int32 index)  
4 [C#]  
5 public int IndexOf(T value, int index)
```

6 Summary

7 Searches for the specified object and returns the zero-based index of the first
8 occurrence within the range of elements in the list that extends from the specified index
9 to the last element.

10 Parameters

Parameter	Description
<i>value</i>	The T to locate in the current list. (The value can be null if T is a reference type.)
<i>index</i>	The zero-based starting index of the search.

11 12 Return Value

13 The zero-based index of the first occurrence of *item* within the range of elements in the
14 list, if found; otherwise, -1.

15 Description

16 The list is searched forward starting at *index* and ending at the last element.
17
18 This method uses the default equality comparer for type T to determine equality of list
19 elements. The default equality comparer for element type T is defined in the Description
20 section of this (class List<T>) specification.
21
22 This method is an O(n) operation, where n is the number of elements from *index* to the
23 end of the list.

24 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>index</i> is less than zero or greater than System.Collections.Generic.List<T>.Count.

1 List<T>.IndexOf(T, System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig int32 IndexOf(!0 value, int32 index, int32 count)  
5 [C#]  
6 public int IndexOf(T value, int index, int count)
```

7 Summary

8 Searches for the specified object and returns the zero-based index of the first
9 occurrence within the range of elements in the list that starts at the specified index and
10 contains the specified number of elements.

11 Parameters

Parameter	Description
<i>value</i>	The T to locate in the current list. (The value can be null if T is a reference type.)
<i>index</i>	The zero-based starting index of the search.
<i>count</i>	The number of elements to search.

12 13 Return Value

14 The zero-based index of the first occurrence of *item* within the specified range of
15 elements in the list, if found; otherwise, -1.

16 Description

17 The list is searched forward starting at *index* and ending at *index + count - 1*, and
18 searching at most *count* terms.

19
20 This method uses the default equality comparer for type T to determine equality of list
21 elements. The default equality comparer for element type T is defined in the Description
22 section of this (class List<T>) specification.

23
24 This method is an O(n) operation, where n is *count*.

25 Exceptions

Exception	Condition
-----------	-----------

System.ArgumentOutOfRangeException

index is less than 0.

-or-

count is less than 0.

-or-

index + *count* is greater than

`System.Collections.Generic.List<T>.Count`.

1

2

1 List<T>.Insert(System.Int32, T) Method

```
2 [ILAsm]  
3 .method public hidebysig void Insert(int32 index, !0 item)  
4 [C#]  
5 public void Insert(int index, T item)
```

6 Summary

7 Inserts an item to the List at the specified position.

8 Parameters

Parameter	Description
<i>index</i>	The zero-based index at which <i>item</i> is to be inserted.
<i>item</i>	The item to insert. (<i>item</i> can be null if T is a reference type.)

9

10 Description

11 System.Collections.Generic.List<T> accepts null as a valid value for reference
12 types and allows duplicate elements.

13

14 If System.Collections.Generic.List<T>.Count already equals
15 System.Collections.Generic.List<T>.Capacity, the capacity of the List is increased.

16

17 If *index* is equal to System.Collections.Generic.List<T>.Count, *item* is added to the
18 end of list.

19 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>index</i> is less than 0.
	-or- <i>index</i> is greater than System.Collections.Generic.List<T>.Count.

20

21

1 List<T>.InsertRange(System.Int32, 2 System.Collections.Generic.IEnumerable<T>) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig void InsertRange(int32 index, class  
6 System.Collections.Generic.IEnumerable`1<!0> collection)  
  
7 [C#]  
8 public void InsertRange(int index, IEnumerable<T> collection)
```

9 Summary

10 Inserts the elements of a collection in the List at the specified position.

11 Parameters

Parameter	Description
<i>index</i>	The zero-based index at which the new elements should be inserted.
<i>collection</i>	The collection whose elements should be inserted into the list. (<i>collection</i> itself cannot be <code>null</code> , but the collection can contain elements that are <code>null</code> , if type <code>T</code> is a reference type.)

12

13 Description

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

16

17 If the new value of `System.Collections.Generic.List<T>.Count` will be greater than
18 `System.Collections.Generic.List<T>.Capacity`, the capacity of the List is increased.

19

20 If *index* is equal to `System.Collections.Generic.List<T>.Count`, the collection is
21 added to the end of list.

22

23 The order of the elements in the collection is preserved in the list.

24 Exceptions

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

System.ArgumentOutOfRangeException

index is less than zero,

-or-

index is greater than
`System.Collections.Generic.List<T>.Count`.

1

2

1 List<T>.LastIndexOf(T) Method

```
2 [ILAsm]  
3 .method public hidebysig int32 LastIndexOf(!0 value)  
4 [C#]  
5 public int LastIndexOf(T value)
```

6 Summary

7 Searches for the specified object and returns the zero-based index of the last occurrence
8 within the entire list.

9 Parameters

Parameter	Description
<i>value</i>	The T to locate in the current list. (The value can be null if T is a reference type.)

10

11 Return Value

12 The zero-based index of the last occurrence of *item* within the entire list, if found;
13 otherwise, -1.

14 Description

15 The list is searched backward starting at the last element and ending at the first
16 element.

17

18 This method uses the default equality comparer for type T to determine equality of list
19 elements. The default equality comparer for element type T is defined in the Description
20 section of this (class List<T>) specification.

21

22 This method is an O(n) operation, where n is
23 System.Collections.Generic.List<T>.Count.

24

1 List<T>.LastIndexOf(T, System.Int32)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig int32 LastIndexOf(!0 value, int32 index)  
  
5 [C#]  
6 public int LastIndexOf(T value, int index)
```

7 Summary

8 Searches for the specified object and returns the zero-based index of the last occurrence
9 within the range of elements in the list that extends from the specified index to the last
10 element.

11 Parameters

Parameter	Description
<i>value</i>	The T to locate in the current list. (The value can be null if T is a reference type.)
<i>index</i>	The zero-based starting index of the search.

12 Return Value

14 The zero-based index of the last occurrence of *item* within the range of elements in the
15 list, if found; otherwise, -1.

16 Description

17 The list is searched backward starting at *index* and ending at the first element.

18 This method uses the default equality comparer for type T to determine equality of list
19 elements. The default equality comparer for element type T is defined in the Description
20 section of this (class List<T>) specification.

22 This method is an O(n) operation, where n is the number of elements from the
23 beginning of the list to *index*.

25 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>index</i> is less than zero or greater than or equal to

	System.Collections.Generic.List<T>.Count.
--	---

1

2

1 List<T>.LastIndexOf(T, System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig int32 LastIndexOf(!0 value, int32 index, int32  
5 count )  
6 [C#]  
7 public int LastIndexOf(T value, int index, int count)
```

8 Summary

9 Searches for the specified object and returns the zero-based index of the last occurrence
10 within the range of elements in the list that starts at the specified index and contains
11 the specified number of elements.

12 Parameters

Parameter	Description
<i>value</i>	The T to locate in the current list. (The value can be null if T is a reference type.)
<i>index</i>	The zero-based starting index of the search.
<i>count</i>	The number of elements to search.

13 14 Return Value

15 The zero-based index of the last occurrence of *item* within the range of elements in the
16 list that contains *count* number of elements and ends at *index*, if found; otherwise, -1.

17 Description

18 The list is searched backward starting at *index* and ending after *count* elements.
19
20 This method uses the default equality comparer for type T to determine equality of list
21 elements. The default equality comparer for element type T is defined in the Description
22 section of this (class List<T>) specification.
23
24 This method is an O(n) operation, where n is *count*.

25 Exceptions

Exception	Condition
-----------	-----------

System.ArgumentOutOfRangeException

index is less than zero, or greater than or equal to

`System.Collections.Generic.List<T>.Count`.

-or-

count is less than 0.

-or-

count is greater than *index* + 1.

1

2

1 List<T>.Remove(T) Method

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

6 Summary

7 Removes the first occurrence of the specified object from the list.

8 Parameters

Parameter	Description
<i>item</i>	The object to be removed from the list.

9

10 Return Value

11 true if *item* is successfully removed; otherwise, false.

12 Description

13 This method uses the default equality comparer for type T to determine equality of list
14 elements. The default equality comparer for element type T is defined in the Description
15 section of this (class List<T>) specification.

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

19

1 List<T>.RemoveAll(System.Predicate<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig int32 RemoveAll(class System.Predicate`1<!0>  
5 match)  
  
6 [C#]  
7 public int RemoveAll(Predicate<T> match)
```

8 Summary

9 Removes the all the elements that match the conditions defined by the specified
10 predicate.

11 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the elements to remove.

12 13 Return Value

14 The number of elements removed from the List.

15 Description

16 The predicate is a delegate that returns `true` if the object passed to it matches the
17 conditions defined in the delegate. The elements of the current List are individually
18 passed to the predicate delegate, and the elements that match the conditions are
19 removed from the List.

20
21 This method is an O(n) operation, where n is
22 `System.Collections.Generic.List<T>.Count`.

23 Exceptions

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

24

25

1 List<T>.RemoveAt(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig void RemoveAt(int32 index)  
4 [C#]  
5 public void RemoveAt(int index)
```

6 Summary

7 Removes the item at the specified index of the list.

8 Parameters

Parameter	Description
<i>index</i>	The zero-based index of the item to remove.

9 10 Description

11 The item is removed and all the elements following it in the List have their indexes
12 reduced by 1.

13
14 This method is an O(n) operation, where n is
15 System.Collections.Generic.List<T>.Count.

16 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>index</i> is less than 0. -or- <i>index</i> is equal to or greater than System.Collections.Generic.List<T>.Count.

17

18

1 List<T>.RemoveRange(System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig void RemoveRange(int32 index, int32 count)  
5 [C#]  
6 public void RemoveRange(int index, int count)
```

7 Summary

8 Removes a range of elements from the list.

9 Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the range of elements to remove.
<i>count</i>	The number of elements to remove.

10

11 Description

12 The items are removed and all the elements following them in the List have their
13 indexes reduced by *count*.

14 Exceptions

Exception	Condition
System.ArgumentException	<i>index</i> + <i>count</i> is greater than System.Collections.Generic.List<T>.Count.
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>count</i> is less than zero.

15

16

1 List<T>.Reverse() Method

```
2 [ILAsm]  
3 .method public hidebysig void Reverse()  
4 [C#]  
5 public void Reverse()
```

6 Summary

7 Reverses the order of the elements in the list.

8 Description

9 This method uses `System.Array.Reverse(System.Array)` to reverse the order of the
10 elements.

11
12 This method is an $O(n)$ operation, where n is
13 `System.Collections.Generic.List<T>.Count`.

14

1 List<T>.Reverse(System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig void Reverse(int32 index, int32 count)  
5 [C#]  
6 public void Reverse(int index, int count)
```

7 Summary

8 Reverses the order of the elements in the specified element range of the list.

9 Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the range of elements to reverse.
<i>count</i>	The number of elements to reverse.

10 11 Description

12 This method reverses the order of the elements in the specified element range

13
14 This method is an O(n) operation, where n is *count*.

15 Exceptions

Exception	Condition
System.ArgumentException	<i>index</i> + <i>count</i> is greater than <code>System.Collections.Generic.List<T>.Count</code> .
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>count</i> is less than zero.

16
17

1 List<T>.Sort() Method

```
2 [ILAsm]  
3 .method public hidebysig void Sort()  
4 [C#]  
5 public void Sort()
```

6 Summary

7 Sorts the elements in the list using the default comparer.

8 Description

9 This method uses the default comparer for type T to determine the order of list
10 elements. If there is no default comparer, then the method throws
11 `System.InvalidOperationException`. The default comparer for a given element type T
12 is defined in the Description section of this (class `List<T>`) specification.

13
14 At worst, this operation is $O(n^2)$, where n is the number of elements to sort. On average
15 it's $O(n \log n)$.

16 Exceptions

Exception	Condition
System.InvalidOperationException	The default comparer cannot find a <code>System.IComparable<T></code> or <code>System.IComparable</code> implementation for type T.

17

18

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

List<T>.Sort(System.Collections.Generic.IComparer<T>) Method

```
[ILAsm]  
.method public hidebysig void Sort(class  
System.Collections.Generic.IComparer`1<!0> comparer)  
  
[C#]  
public void Sort(IComparer<T> comparer)
```

Summary

Sorts the elements in the list using the specified comparer.

Parameters

Parameter	Description
<i>comparer</i>	The System.Collections.Generic.IComparer<T> implementation to use when comparing elements. -or- null to use the default comparer.

Description

If the given comparer is non-null, it is used to determine the order of list elements. If the given comparer is null, the default comparer for type T is used; if there is no default comparer, then the method throws System.InvalidOperationException. The default comparer for a given element type T is defined in the Description section of this (class List<T>) specification.

At worst, this operation is O(n²), where n is the number of elements to sort. On average it's O(n log n).

Exceptions

Exception	Condition
System.InvalidOperationException	<i>comparer</i> is null, and the default comparer cannot find a System.IComparable<T> or System.IComparable implementation for type T.

1 List<T>.Sort(System.Int32, System.Int32, 2 System.Collections.Generic.IComparer<T>) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig void Sort(int32 index, int32 count, class  
6 System.Collections.Generic.IComparer`1<!0> comparer)  
7  
8 [C#]  
9 public void Sort(int index, int count, IComparer<T> comparer)
```

9 Summary

10 Sorts the elements in the list using the specified comparer.

11 Parameters

Parameter	Description
<i>index</i>	The zero-based starting index of the range of elements to sort.
<i>count</i>	The number of elements to sort.
<i>comparer</i>	The <code>System.Collections.Generic.IComparer<T></code> implementation to use when comparing elements. -or- null to use the default comparer.

12

13 Description

14 If the given comparer is non-null, it is used to determine the order of list elements. If
15 the given comparer is null, the default comparer for type T is used; if there is no
16 default comparer, then the method throws `System.InvalidOperationException`. The
17 default comparer for a given element type T is defined in the Description section of this
18 (class `List<T>`) specification.

19

20 At worst, this operation is $O(n^2)$, where n is the number of elements to sort. On average
21 it's $O(n \log n)$.

22 Exceptions

Exception	Condition
-----------	-----------

System.ArgumentException	<i>index</i> + <i>count</i> is greater than System.Collections.Generic.List<T>.Count.
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>count</i> is less than zero.
System.InvalidOperationException	<i>comparer</i> is null, and the default comparer cannot find a System.IComparable<T> or System.IComparable implementation for type T.

1

2

1 List<T>.Sort(System.Comparison<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig void Sort(class System.Comparison`1<!0>  
5 comparison)  
  
6 [C#]  
7 public void Sort(Comparison<T> comparison)
```

8 Summary

9 Sorts the elements in the list using the specified comparison.

10 Parameters

Parameter	Description
<i>comparison</i>	The comparison to use when comparing elements.

11 Description

12 At worst, this operation is $O(n^2)$, where n is the number of elements to sort. On average
13 it's $O(n \log n)$.

15 Exceptions

Exception	Condition
System.ArgumentNullException	<i>comparison</i> is null.

16
17

1

2 List<T>.System.Collections.Generic.IEnumera 3 ble<T>.GetEnumerator() Method

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

10 Summary

11 This method is implemented to support the
12 System.Collections.Generic.IEnumerable<T> interface.

13

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

4 `[ILAsm]`
5 `.method private hidebysig virtual final void`
6 `System.Collections.ICollection.CopyTo(class System.Array array, int32`
7 `index)`

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

10 **Summary**

11 This method is implemented to support the System.Collections.ICollection
12 interface.

13

1
2 **List<T>.System.Collections.IEnumerable.Get**
3 **Enumerator() Method**

```
4 [ILAsm]  
5 .method private hidebysig virtual final class  
6 System.Collections.IEnumerator  
7 System.Collections.IEnumerable.GetEnumerator()  
  
8 [C#]  
9 IEnumerator IEnumerable.GetEnumerator()
```

10 **Summary**

11 This method is implemented to support the System.Collections.IEnumerable
12 interface.

13

1
2 **List<T>.System.Collections.IList.Add(System.**
3 **Object) Method**

```
4 [ILAsm]  
5 .method private hidebysig virtual final int32  
6 System.Collections.IList.Add(object value)  
7 [C#]  
8 int IList.Add(object value)
```

9 **Summary**

10 This method is implemented to support the System.Collections.IList interface.

11

1
2 **List<T>.System.Collections.IList.Contains(Sy**
3 **stem.Object) Method**

```
4 [ILAsm]  
5 .method private hidebysig virtual final bool  
6 System.Collections.IList.Contains(object value)  
7  
8 [C#]  
9 bool IList.Contains(object value)
```

9 **Summary**

10 This method is implemented to support the System.Collections.IList interface.

11

1
2 **List<T>.System.Collections.IList.IndexOf(System.Object) Method**
3

```
4 [ILAsm]  
5 .method private hidebysig virtual final int32  
6 System.Collections.IList.IndexOf(object value)  
  
7 [C#]  
8 int IList.IndexOf(object value)
```

9 **Summary**

10 This method is implemented to support the System.Collections.IList interface.

11

1
2 **List<T>.System.Collections.IList.Insert(System.Int32, System.Object) Method**
3

```
4 [ILAsm]  
5 .method private hidebysig virtual final void  
6 System.Collections.IList.Insert(int32 index, object value)  
7  
8 [C#]  
9 void IList.Insert(int index, object value)
```

9 **Summary**

10 This method is implemented to support the System.Collections.IList interface.

11

1
2 **List<T>.System.Collections.IList.Remove(System.Object) Method**
3

```
4 [ILAsm]  
5 .method private hidebysig virtual final void  
6 System.Collections.IList.Remove(object value)  
7  
8 [C#]  
9 void IList.Remove(object value)
```

9 **Summary**

10 This method is implemented to support the System.Collections.IList interface.

11

1
2 **List<T>.System.Collections.IList.RemoveAt(S**
3 **ystem.Int32) Method**

```
4 [ILAsm]  
5 .method private hidebysig virtual final void  
6 System.Collections.IList.RemoveAt(int32 index)  
  
7 [C#]  
8 void IList.RemoveAt(int index)
```

9 **Summary**

10 This method is implemented to support the System.Collections.IList interface.

11

1 List<T>.ToArray() Method

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

6 Summary

7 Copies the elements in the list to a new array.

8 Return Value

9 The new array containing a copy of the list's elements.

10 Description

11 This an O(n) operation, where n is `System.Collections.Generic.List<T>.Count`.

12

1 List<T>.TrimExcess() Method

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

6 Summary

7 Suggests that the capacity be reduced to the actual number of elements in the list.

8 Description

9 This method can be used to suggest a collection's memory overhead be minimized, e.g.,
10 if no new elements are expected to be added to the collection.

11
12 [*Note:* To reset a list to its initial state, call the
13 System.Collections.Generic.List.Clear method before calling
14 System.Collections.Generic.List.TrimExcess.]
15
16

17

1 List<T>.TrimToSize() Method

```
2 [ILAsm]  
3 method public hidebysig void TrimToSize()  
4 [C#]  
5 public void TrimToSize()
```

6 Summary

7 Sets the capacity to the actual number of elements in the list.

8 Description

9 This method can be used to minimize a list's memory overhead if no new elements are
10 expected to be added to the list.

11
12 To reset a List to its initial state, call the `System.Collections.Generic.List<T>.Clear`
13 `method` before calling `System.Collections.Generic.List<T>.TrimToSize`. Trimming
14 an empty list sets the capacity of the list to the default capacity.

15

1 List<T>.TrueForAll(System.Predicate<T>)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig bool TrueForAll(class System.Predicate`1<!0>  
5 match)  
  
6 [C#]  
7 public bool TrueForAll(Predicate<T> match)
```

8 Summary

9 Determines whether every element in the List matches the conditions defined by the
10 specified predicate.

11 Parameters

Parameter	Description
<i>match</i>	The predicate delegate that specifies the check against the elements.

12 Return Value

14 *true*, if every element in the List matches the conditions defined by the specified
15 predicate; otherwise, *false*.

16 Description

17 The predicate is a delegate that returns *true* if the object passed to it matches the
18 conditions defined in the delegate. The elements of the current List are individually
19 passed to the predicate delegate. The elements are processed sequentially and on the
20 same thread.

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>match</i> is null.

22
23

1 List<T>.Capacity Property

```
2 [ILAsm]
3 .property int32 Capacity { public hidebysig specialname int32
4 get_Capacity() public hidebysig specialname void set_Capacity(int32 value)
5 }
6 [C#]
7 public int Capacity { get; set; }
```

8 Summary

9 Gets or sets the number of elements the current instance can contain.

10 Property Value

11 A System.Int32 containing the number of elements the current instance can contain.

12 Description

13 This property is read/write.

14
15 System.Collections.Generic.List<T>.Capacity is the number of elements that the
16 list is capable of storing without needing to be extended.
17 System.Collections.Generic.List<T>.Count is the number of elements that are
18 actually in the list.

19
20 System.Collections.Generic.List<T>.Capacity is always greater than or equal to
21 System.Collections.Generic.List<T>.Count. When
22 System.Collections.Generic.List<T>.Count exceeds
23 System.Collections.Generic.List<T>.Capacity while adding elements, the capacity
24 is increased.

25
26 The capacity can be decreased by calling
27 System.Collections.Generic.List<T>.TrimToSize or by setting the
28 System.Collections.Generic.List<T>.Capacity property explicitly.

29 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	Attempt to set the capacity to a value less than System.Collections.Generic.List<T>.Count.

30
31

1 List<T>.Count Property

```
2 [ILAsm]  
3 .property int32 Count { public hidebysig specialname instance int32  
4 get_Count () }  
5 [C#]  
6 public int Count { get; }
```

7 Summary

8 Gets the number of elements contained in the current instance.

9 Property Value

10 The number of elements in the current instance.

11 Description

12 This property is read-only.

13

1 List<T>.Item Property

```
2 [ILAsm]  
3 .property object Item(int32 index) { public hidebysig specialname !0  
4 get_Item(int32 index) public hidebysig specialname void set_Item(int32  
5 index, !0 value) }  
  
6 [C#]  
7 public T this[int index] { get; set; }
```

8 Summary

9 Gets or sets the element at the specified index of the current instance.

10 Parameters

Parameter	Description
<i>index</i>	The zero-based index of the element in the current instance to get or set.

11 12 Property Value

13 The element at the specified index of the current instance.

14 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>index</i> < 0. -or- <i>index</i> >= System.Collections.Generic.List<T>.Count of the current instance.

15

16

1
2 **List<T>.System.Collections.Generic.ICollection**
3 **n<T>.IsReadOnly Property**

```
4 [ILAsm]  
5 .property bool System.Collections.Generic.ICollection`1<T>.IsReadOnly {  
6 private hidebysig virtual final specialname bool get_IsReadOnly() }  
7 [C#]  
8 bool ICollection<T>.IsReadOnly { get; }
```

9 **Summary**

10 This read-only property is implemented to support the
11 System.Collections.Generic.ICollection<T> interface.

12
13 [*Note:* For more information, see
14 System.Collections.Generic.ICollection<T>.IsReadOnly.

15
16]

17

1
2 **List<T>.System.Collections.ICollection.IsSyn**
3 **chronized Property**

```
4 [ILAsm]  
5 .property bool System.Collections.ICollection.IsSynchronized { private  
6 hidebyref virtual final specialname bool get_IsSynchronized() }  
7 [C#]  
8 bool ICollection.IsSynchronized { get; }
```

9 **Summary**

10 This read-only property is implemented to support the
11 System.Collections.ICollection interface.

12

1
2 **List<T>.System.Collections.ICollection.SyncR**
3 **oot Property**

```
4 [ILAsm]  
5 .property object System.Collections.ICollection.SyncRoot { private  
6 hidebysig virtual final specialname object get_SyncRoot() }  
7  
8 [C#]  
9 object ICollection.SyncRoot { get; }
```

9 **Summary**

10 This read-only property is implemented to support the
11 System.Collections.ICollection interface.

12

1 List<T>.System.Collections.IList.IsFixedSize 2 Property

```
3 [ILAsm]  
4 .property bool System.Collections.IList.IsFixedSize { private hidebysig  
5 virtual final specialname bool get_IsFixedSize() }  
  
6 [C#]  
7 bool IList.IsFixedSize { get; }
```

8 Summary

9 This read-only property is implemented to support the System.Collections.IList
10 interface.

11

1 List<T>.System.Collections.IList.IsReadOnly 2 Property

```
3 [ILAsm]  
4 .property bool System.Collections.IList.IsReadOnly { private hideby sig  
5 virtual final specialname bool get_IsReadOnly() }  
  
6 [C#]  
7 bool IList.IsReadOnly { get; }
```

8 Summary

9 This read-only property is implemented to support the System.Collections.IList
10 interface.

11

1 List<T>.System.Collections.IList.Item 2 Property

```
3 [ILAsm]  
4 .property object System.Collections.IList.Item(int32 index) { private  
5 hidebysig virtual final specialname object get_Item(int32 index) private  
6 hidebysig virtual final specialname void set_Item(int32 index, object  
7 value) }  
  
8 [C#]  
9 object IList.this[int index] { get; set; }
```

10 Summary

11 This read-only property is implemented to support the System.Collections.IList
12 interface.

13