System.Collections.Generic.List<T> Class

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

[C#]
public class List<T>: IList<T>, ICollection<T>, IEnumerable<T>, IList,
ICollection, IEnumerable
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
  - CLSCompliantAttribute(true)

Implements:

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

Summary

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

Inherits From: System.Object

Library: BCL

Thread Safety: Static members of this type are thread-safe. Any instance members are not guaranteed to be thread safe. A list can support multiple readers concurrently, as long as the collection is not modified. Even so, enumerating through a collection is intrinsically not a thread safe procedure. [Note: 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.]

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.

List<T>() Constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor()

[C#]
public List()
```

Summary

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

Description

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

List<T>(System.Collections.Generic.IEnumerable<T>) Constructor

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

[C#]
public List(IEnumerable<T> collection)
```

Summary

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

Parameters

Parameter	Description
collection	The collection from which to copy the elements.

Description

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

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

Exceptions

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

List<T>(System.Int32) Constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor(int32 capacity)

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

Summary

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

Parameters

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

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	*capacity* is less than zero.

List<T>.Add(T) Method

```
[ILAsm]
.method public hidebysig void Add(!0 item)

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

Summary

Adds an item to the end of the list.

Parameters

Parameter	Description
item	The item to add to the end of the list. (*item* can be `null` if T is a reference type.)

Description

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

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

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

List<T>.AddRange(System.Collections.Generic.IEnumerable<T>) Method

```
[ILAsm]
.method public hidebysig void AddRange(class
System.Collections.Generic.IEnumerable`1<!0> collection)

[C#]
public void AddRange(IEnumerable<T> collection)
```

Summary

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

Parameters

Parameter	Description
collection	The collection whose elements are added to the end of the list.

Description

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

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

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

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

Exceptions

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

1
2

3

List<T>.AsReadOnly() Method

```
[ILAsm]
.method public hidebysig class System.Collections.Generic.IList`1<!0>
AsReadOnly()

[C#]
public IList<T> AsReadOnly()
```

Summary

Returns a read-only wrapper to the current List.

Return Value

A read-only wrapper for the current List.

Description

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

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

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
item	The element for which to search. (*item* 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.

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig int32 BinarySearch(!0 item, class
System.Collections.Generic.IComparer`1<!0> comparer)

[C#]
public int BinarySearch(T item, IComparer<T> comparer)
```

Summary

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

Parameters

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

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

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.

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

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 System.Collections.Generic.List<T> 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.

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

Exceptions

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

List<T>.BinarySearch(System.Int32, System.Int32, T, System.Collections.Generic.IComparer<T>) Method

```
[ILAsm]
.method public hidebysig int32 BinarySearch(int32 index, int32 count, !0
item, class System.Collections.Generic.IComparer`1<!0> comparer)

[C#]
public int BinarySearch(int index, int count, T item, IComparer<T>
comparer)
```

Summary

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

Parameters

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

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 *index* + *count*.

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.

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

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 System.Collections.Generic.List<T> 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.

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

Exceptions

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

	T.

List<T>.Clear() Method

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

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

Summary

Removes all elements from the list.

Description

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

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

]

List<T>.Contains(T) Method

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

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

Summary

Determines whether the list contains a specific value.

Parameters

Parameter	Description
item	The object to locate in the current collection. (*item* can be `null` if T is a reference type.)

Return Value

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

Description

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

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

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

```
[ILAsm]
.method public hidebysig System.Collections.Generic.List`1<!1>
ConvertAll<U>(class System.Converter`2<!0,!1> converter)

[C#]
public List<U> ConvertAll<U>(Converter<T,U> converter)
```

Summary

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

Parameters

Parameter	Description
converter	A converter delegate that converts each element from one type to another type.

Return Value

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

Description

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

The current List remains unchanged.

Exceptions

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

1
2
3

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

```
[ILAsm]
.method public hidebysig void CopyTo(!0[] array)

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

Summary

Copies the entire list to an array.

Parameters

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

Description

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

Exceptions

Exception	Condition
System.ArgumentException	*array* is multidimensional. -or- *array* does not have zero-based indexing. -or- The number of elements in the list is greater than the number of elements that the destination *array* can contain. -or-

	Type T is not assignable to the element type of the destination array.
System.ArgumentNullException	*array* is `null`.

1
2

3

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

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

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

Summary

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

Parameters

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

Description

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

Exceptions

Exception	Condition
System.ArgumentException	*array* is multidimensional. -or- *array* does not have zero-based indexing. -or- The sum of *arrayIndex* 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	*array* is `null`.
System.ArgumentOutOfRangeException	*arrayIndex* is less than zero.

1
2

3

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

```
[ILAsm]
.method public hidebysig void CopyTo(int32 index,!0[] array, int32
arrayIndex, int32 count)

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

Summary

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

Parameters

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

Description

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

Exceptions

Exception	Condition
System.ArgumentException	*array* is multidimensional. -or-

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

1
2

3

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

```
[ILAsm]
.method public hidebysig bool Exists(class System.Predicate`1<!0> match)

[C#]
public bool Exists(Predicate<T> match)
```

Summary

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

Parameters

Parameter	Description
match	The predicate delegate that specifies the elements to search for.

Return Value

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

Description

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, and processing is stopped when a match is found.

Exceptions

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

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

```
[ILAsm]
.method public hidebysig !0 Find(class System.Predicate`1<!0> match)

[C#]
public T Find(Predicate<T> match)
```

Summary

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

Parameters

Parameter	Description
match	The predicate delegate that specifies the element to search for.

Return Value

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

Description

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, moving forward in the List, starting with the first element and ending with the last element. Processing is stopped when a match is found.

Exceptions

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

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

```
[ILAsm]
.method public hidebysig System.Collections.Generic.List`1<!0>
FindAll(class System.Predicate<!0> match)

[C#]
public List<T> FindAll(Predicate<T> match)
```

Summary

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

Parameters

Parameter	Description
match	The predicate delegate that specifies the elements to search for.

Return Value

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

Description

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, and the elements that match the conditions are saved in the returned List.

Exceptions

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

1

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

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

[C#]
public int FindIndex(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 first occurrence within the List.

Parameters

Parameter	Description
match	The predicate delegate that specifies the element to search for.

Return Value

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

Description

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

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	*match* is `null`.

1
2
3

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

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

[C#]
public int FindIndex(int index, 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 first occurrence within the range of elements in the List that extends from the specified index to the last element.

Parameters

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

Return Value

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

Description

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

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	*match* is `null`.
System.ArgumentOutOfRangeException	*index* is less than 0 or greater than or equal to `System.Collections.Generic.List<T>.Count`.

1
2
3

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

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

[C#]
public int FindIndex(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 first occurrence within the range of elements in the List that starts at the specified index and contains the specified number of elements.

Parameters

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

Return Value

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

Description

The List is searched forward 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	*match* 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`.

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

```
[ILAsm]
.method public hidebysig !0 FindLast(class System.Predicate`1<!0> match)

[C#]
public T FindLast(Predicate<T> match)
```

Summary

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

Parameters

Parameter	Description
match	The predicate delegate that specifies the element to search for.

Return Value

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

Description

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, moving backward in the List, starting with the last element and ending with the first element. Processing is stopped when a match is found.

Exceptions

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

1

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

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

[C#]
public int FindLastIndex(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 List.

Parameters

Parameter	Description
match	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 the last element and ending at the first element.

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	*match* is `null`.

1
2

3

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

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

[C#]
public int FindLastIndex(int index, 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 extends from the specified index to the first element.

Parameters

Parameter	Description
index	The zero-based starting index of the backward search.
match	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 at the first element.

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	*match* is `null`.
System.ArgumentOutOfRangeException	*index* is less than 0 or greater than or equal to `System.Collections.Generic.List<T>.Count`.

1
2
3

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
index	The zero-based starting index of the search.
count	The number of elements to search.
match	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	*match* 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

3

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

```
[ILAsm]
.method public hidebysig void ForEach(class System.Action`1<!0> action)

[C#]
public void ForEach(Action<T> action)
```

Summary

Performs the specified action on each element of the List.

Parameters

Parameter	Description
action	The action delegate to perform on each element of the List.

Description

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

Exceptions

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

List<T>.GetEnumerator() Method

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

[C#]
public List<T>.Enumerator GetEnumerator()
```

Summary

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

Return Value

An enumerator for the list.

Usage

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

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

```
[ILAsm]
.method public hidebysig class System.Collections.Generic.List`1<!0>
GetRange(int32 index, int32 count)

[C#]
public List<T> GetRange(int index, int count)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition
System.ArgumentException	*index* + *count* is greater than

	`System.Collections.Generic.List<T>.Count`.
System.ArgumentOutOfRangeException	*index* is less than 0. -or- *count* is less than 0.

1
2
3

List<T>.IndexOf(T) Method

```
[ILAsm]
.method public hidebysig int32 IndexOf(!0 value)

[C#]
public int IndexOf(T value)
```

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

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

```
[ILAsm]
.method public hidebysig int32 IndexOf(!0 value, int32 index)

[C#]
public int IndexOf(T value, int index)
```

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

Exceptions

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

1
2

3

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

```
[ILAsm]
.method public hidebysig int32 IndexOf(!0 value, int32 index, int32 count)

[C#]
public int IndexOf(T value, int index, int count)
```

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

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`.

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

```
[ILAsm]
.method public hidebysig void Insert(int32 index, !0 item)

[C#]
public void Insert(int index, T item)
```

Summary

Inserts an item to the List at the specified position.

Parameters

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

Description

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

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

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

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	*index* is less than 0. -or- *index* is greater than

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

1
2
3

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

```
[ILAsm]
.method public hidebysig void InsertRange(int32 index, class
System.Collections.Generic.IEnumerable`1<!0> collection)

[C#]
public void InsertRange(int index, IEnumerable<T> collection)
```

Summary

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

Parameters

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

Description

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

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

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*collection* is `null`.
System.ArgumentOutOfRangeException	*index* is less than zero, -or- *index* is greater than `System.Collections.Generic.List<T>.Count`.

1
2
3

List<T>.LastIndexOf(T) Method

```
[ILAsm]
.method public hidebysig int32 LastIndexOf(!0 value)

[C#]
public int LastIndexOf(T value)
```

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

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

```
[ILAsm]
.method public hidebysig int32 LastIndexOf(!0 value, int32 index)

[C#]
public int LastIndexOf(T value, int index)
```

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	*index* is less than zero or greater than or equal to `System.Collections.Generic.List<T>.Count`.

1

2

3

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

```
[ILAsm]
.method public hidebysig int32 LastIndexOf(!0 value, int32 index, int32
count)

[C#]
public int LastIndexOf(T value, int index, int count)
```

Summary

Searches for the specified object 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.

Parameters

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

Return Value

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

Description

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

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

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

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.

List<T>.Remove(T) Method

```
[ILAsm]
.method public hidebysig bool Remove(!0 item)

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

Summary

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

Parameters

Parameter	Description
item	The object to be removed from the list.

Return Value

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

Description

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

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

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

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

[C#]
public int RemoveAll(Predicate<T> match)
```

Summary

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

Parameters

Parameter	Description
match	The predicate delegate that specifies the elements to remove.

Return Value

The number of elements removed from the List.

Description

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, and the elements that match the conditions are removed from the List.

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

Exceptions

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

1
2
3

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

```
[ILAsm]
.method public hidebysig void RemoveAt(int32 index)

[C#]
public void RemoveAt(int index)
```

Summary

Removes the item at the specified index of the list.

Parameters

Parameter	Description
index	The zero-based index of the item to remove.

Description

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig void RemoveRange(int32 index, int32 count)

[C#]
public void RemoveRange(int index, int count)
```

Summary

Removes a range of elements from the list.

Parameters

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

Description

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

Exceptions

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

1
2
3

List<T>.Reverse() Method

```
[ILAsm]
.method public hidebysig void Reverse()

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

Summary

Reverses the order of the elements in the list.

Description

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

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

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

```
[ILAsm]
.method public hidebysig void Reverse(int32 index, int32 count)

[C#]
public void Reverse(int index, int count)
```

Summary

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

Parameters

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

Description

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

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

Exceptions

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

1
2
3

List<T>.Sort() Method

```
[ILAsm]
.method public hidebysig void Sort()

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

Summary

Sorts the elements in the list using the default comparer.

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.

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

Exceptions

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

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
comparer	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 $O(n^2)$, where n is the number of elements to sort. On average it's O(n log n).

Exceptions

Exception	Condition

System.InvalidOperationException	*comparer* is `null`, and the default comparer cannot find a `System.IComparable<T>` or `System.IComparable` implementation for type `T`.

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

```
[ILAsm]
.method public hidebysig void Sort(int32 index, int32 count, class
System.Collections.Generic.IComparer`1<!0> comparer)

[C#]
public void Sort(int index, int count, IComparer<T> comparer)
```

Summary

Sorts the elements in the list using the specified comparer.

Parameters

Parameter	Description
index	The zero-based starting index of the range of elements to sort.
count	The number of elements to sort.
comparer	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 $O(n^2)$, where n is the number of elements to sort. On average it's O(n log n).

Exceptions

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

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

```
[ILAsm]
.method public hidebysig void Sort(class System.Comparison`1<!0>
comparison)

[C#]
public void Sort(Comparison<T> comparison)
```

Summary

Sorts the elements in the list using the specified comparison.

Parameters

Parameter	Description
comparison	The comparison to use when comparing elements.

Description

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

Exceptions

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

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

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

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

Summary

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

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

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

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

Summary

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

List<T>.System.Collections.IEnumerable.GetEnumerator() Method

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

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

Summary

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

List<T>.System.Collections.IList.Add(System.Object) Method

```
[ILAsm]
.method private hidebysig virtual final int32
System.Collections.IList.Add(object value)

[C#]
int IList.Add(object value)
```

Summary

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

List<T>.System.Collections.IList.Contains(System.Object) Method

```
[ILAsm]
.method private hidebysig virtual final bool
System.Collections.IList.Contains(object value)

[C#]
bool IList.Contains(object value)
```

Summary

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

List<T>.System.Collections.IList.IndexOf(System.Object) Method

```
[ILAsm]
.method private hidebysig virtual final int32
System.Collections.IList.IndexOf(object value)

[C#]
int IList.IndexOf(object value)
```

Summary

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

List<T>.System.Collections.IList.Insert(System.Int32, System.Object) Method

```
[ILAsm]
.method private hidebysig virtual final void
System.Collections.IList.Insert(int32 index, object value)

[C#]
void IList.Insert(int index, object value)
```

Summary

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

List<T>.System.Collections.IList.Remove(System.Object) Method

```
[ILAsm]
.method private hidebysig virtual final void
System.Collections.IList.Remove(object value)

[C#]
void IList.Remove(object value)
```

Summary

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

List<T>.System.Collections.IList.RemoveAt(System.Int32) Method

```
[ILAsm]
.method private hidebysig virtual final void
System.Collections.IList.RemoveAt(int32 index)

[C#]
void IList.RemoveAt(int index)
```

Summary

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

List<T>.ToArray() Method

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

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

Summary

Copies the elements in the list to a new array.

Return Value

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

Description

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

List<T>.TrimExcess() Method

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

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

Summary

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

Description

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

[*Note:* To reset a list to its initial state, call the `System.Collections.Generic.List.Clear` method before calling `System.Collections.Generic.List.TrimExcess`.]

List<T>.TrimToSize() Method

```
[ILAsm]
method public hidebysig void TrimToSize()

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

Summary

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

Description

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

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

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

```
[ILAsm]
.method public hidebysig bool TrueForAll(class System.Predicate`1<!0>
match)

[C#]
public bool TrueForAll(Predicate<T> match)
```

Summary

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

Parameters

Parameter	Description
match	The predicate delegate that specifies the check against the elements.

Return Value

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

Description

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. The elements are processed sequentially and on the same thread.

Exceptions

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

1
2
3

List<T>.Capacity Property

```
[ILAsm]
.property int32 Capacity { public hidebysig specialname int32
get_Capacity() public hidebysig specialname void set_Capacity(int32 value)
}

[C#]
public int Capacity { get; set; }
```

Summary

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

Property Value

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

Description

This property is read/write.

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

`System.Collections.Generic.List<T>.Capacity` is always greater than or equal to `System.Collections.Generic.List<T>.Count`. When `System.Collections.Generic.List<T>.Count` exceeds `System.Collections.Generic.List<T>.Capacity` while adding elements, the capacity is increased.

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.

Exceptions

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

1
2
3

List<T>.Count Property

```
[ILAsm]
.property int32 Count { public hidebysig specialname instance int32
get_Count () }

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

Summary

Gets the number of elements contained in the current instance.

Property Value

The number of elements in the current instance.

Description

This property is read-only.

List<T>.Item Property

```
[ILAsm]
.property object Item(int32 index) { public hidebysig specialname !0
get_Item(int32 index) public hidebysig specialname void set_Item(int32
index, !0 value) }

[C#]
public T this[int index] { get; set; }
```

Summary

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

Parameters

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

Property Value

The element at the specified index of the current instance.

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	*index* < 0. -or- *index* >= `System.Collections.Generics.List<T>.Count` of the current instance.

List<T>.System.Collections.Generic.ICollection<T>.IsReadOnly Property

```
[ILAsm]
.property bool System.Collections.Generic.ICollection`1<T>.IsReadOnly {
private hidebysig virtual final specialname bool get_IsReadOnly() }

[C#]
bool ICollection<T>.IsReadOnly { get; }
```

Summary

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

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

]

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

```
[ILAsm]
.property bool System.Collections.ICollection.IsSynchronized { private
hidebysig virtual final specialname bool get_IsSynchronized() }

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

Summary

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

List<T>.System.Collections.ICollection.SyncRoot Property

```
[ILAsm]
.property object System.Collections.ICollection.SyncRoot { private
hidebysig virtual final specialname object get_SyncRoot() }

[C#]
object ICollection.SyncRoot { get; }
```

Summary

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

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

```
[ILAsm]
.property bool System.Collections.IList.IsFixedSize { private hidebysig
virtual final specialname bool get_IsFixedSize() }

[C#]
bool IList.IsFixedSize { get; }
```

Summary

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

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

```
[ILAsm]
.property bool System.Collections.IList.IsReadOnly { private hidebysig
virtual final specialname bool get_IsReadOnly() }

[C#]
bool IList.IsReadOnly { get; }
```

Summary

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

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

```
[ILAsm]
.property object System.Collections.IList.Item(int32 index) { private
hidebysig virtual final specialname object get_Item(int32 index) private
hidebysig virtual final specialname void set_Item(int32 index, object
value) }

[C#]
object IList.this[int index] { get; set; }
```

Summary

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