

1 System.Array Class

```
2 [ILAsm]
3 .class public abstract serializable Array extends System.Object implements
4 System.ICloneable, System.Collections.ICollection,
5 System.Collections.IEnumerable, System.Collections.IList
6
7 [C#]
8 public abstract class Array: ICloneable, ICollection, IEnumerable, IList
```

8 Assembly Info:

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

14 Implements:

- 15 • **System.ICloneable**
- 16 • **System.Collections.ICollection**
- 17 • **System.Collections.IEnumerable**
- 18 • **System.Collections.IList**

19 Summary

20 Serves as the base class for arrays. Provides methods for creating, copying,
21 manipulating, searching, and sorting arrays.

22 Inherits From: System.Object

23
24 **Library:** BCL

25
26 **Thread Safety:** All public static members of this type are safe for multithreaded operations.
27 No instance members are guaranteed to be thread safe.

29 Description

30 This class is intended to be used as a base class by language implementations that
31 support arrays. Only the system can derive from this type: derived classes of
32 `System.Array` are not to be created by the developer.

33
34 [*Note:* An array is a collection of identically typed data *elements* that are accessed and
35 referenced by sets of integral *indices*.

36
37 The *rank* of an array is the number of dimensions in the array. Each dimension has its
38 own set of indices. An array with a rank greater than one can have a different lower
39 bound and a different number of elements for each dimension. Multidimensional arrays

1 (i.e. arrays with a rank greater than one) are processed in row-major order.
2
3 The *lower bound* of a dimension is the starting index of that dimension.
4
5 The *length* of an array is the total number of elements contained in all of its dimensions.
6
7 A *vector* is a one-dimensional array with a *lower bound* of '0'.
8
9 If the implementer creates a derived class of `System.Array`, expected `System.Array`
10 behavior cannot be guaranteed. For information on array-like objects with increased
11 functionality, see the `System.Collections.IList` and
12 `System.Collections.Generic.IList<T>` interfaces. For more information regarding the
13 use of arrays versus the use of collections, see Partition V of the CLI Specification.
14
15]

16
17 Every specific `System.Array` type has three instance methods defined on it. While some
18 programming languages allow direct access to these methods, they are primarily
19 intended to be called by the output of compilers based on language syntax that deals
20 with arrays.

- 21 • `Get`: Takes as many `System.Int32` arguments as the array has dimensions and
22 returns the value stored at the given index. It throws a
23 `System.IndexOutOfRangeException` exception for invalid indices.
- 24 • `Set`: Takes as many `System.Int32` arguments as the array has dimensions, plus one
25 additional argument (the last argument) which has the same type as an array
26 element. It stores the final value in the specified index of the array. It throws a
27 `System.IndexOutOfRangeException` exception for invalid indices.
- 28 • `Address`: Takes as many `System.Int32` arguments as the array has dimensions and
29 returns the address of the element at the given index. It throws a
30 `System.IndexOutOfRangeException` exception for invalid indices.

31 In addition, every specific `System.Array` type has a constructor on it that takes as many
32 non-negative `System.Int32` arguments as the array has dimensions. The arguments specify
33 the number of elements in each dimension, and a lower bound of 0. Thus, a two-
34 dimensional array of `System.Int32` objects would have a constructor that could be called
35 with (2, 4) as its arguments to create an array of eight zeros with the first dimension
36 indexed with 0 and 1 and the second dimension indexed with 0, 1, 2, and 3.

37
38 For all specific array types except vectors (i.e. those permitted to have non-zero lower
39 bounds and those with more than one dimension) there is an additional constructor. It takes
40 twice as many arguments as the array has dimensions. The arguments are considered in
41 pairs, with the first of the pair specifying the lower bound for that dimension and the second
42 specifying the total number of elements in that dimension. Thus, a two-dimensional array of
43 `System.Int32` objects would also have a constructor that could be called with (-1, 2, 1,
44 3) as its arguments, specifying an array of 6 zeros, with the first dimension indexed by -1
45 and 0, and the second dimension indexed by 1, 2, and 3.

46

1 Enumeration over an array occurs in ascending row-major order, starting from the first
2 element. (For example, a 2x3 array is traversed in the order [0,0], [0,1], [0,2], [1,0],
3 [1,1], and [1,2].)

4

5 Parallel implementation of methods taking a `System.Predicate` argument are not
6 permitted.

7

1 Array() Constructor

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

6 Summary

7 Constructs a new instance of the `System.Array` class.

8

1 Array.AsReadOnly<T>(T[]) Method

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

7 Summary

8 Returns a read-only `System.Collections.Generic.IList<T>` wrapper around the
9 specified array.

10 Parameters

Parameter	Description
<i>array</i>	The array to wrap in a read-only <code>System.Collections.Generic.IList<T></code> wrapper.

11 12 Return Value

13 A read-only `System.Collections.Generic.IList<T>` wrapper around the specified
14 array.

15 Description

16 [Note: To prevent any modifications to the array, expose the array only through this
17 wrapper.]

18
19
20
21 The returned `IList<T>` has the same enumeration order as the array it wraps.
22
23 A collection that is read-only is simply a collection with a wrapper that prevents
24 modifying the underlying array; therefore, if changes are made to the underlying array,
25 the read-only collection reflects those changes.

26 Exceptions

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

27

28

1 Array.BinarySearch(System.Array, 2 System.Int32, System.Int32, System.Object, 3 System.Collections.IComparer) Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 BinarySearch(class System.Array  
6 array, int32 index, int32 length, object value, class  
7 System.Collections.IComparer comparer)  
  
8 [C#]  
9 public static int BinarySearch(Array array, int index, int length, object  
10 value, IComparer comparer)
```

11 Summary

12 Searches the specified section of the specified one-dimensional System.Array for the
13 specified value, using the specified System.Collections.IComparer implementation.

14 Parameters

Parameter	Description
<i>array</i>	A System.Array to search.
<i>index</i>	A System.Int32 that contains the index at which searching starts.
<i>length</i>	A System.Int32 that contains the number of elements to search, beginning with <i>index</i> .
<i>value</i>	A System.Object for which to search.
<i>comparer</i>	The System.Collections.IComparer implementation to use when comparing elements. Specify a null reference to use the System.IComparable implementation of each element.

15 16 Return Value

17 A System.Int32 with one of the following values based on the result of the search
18 operation.

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.

The bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found, and at least one array element in the range of <i>index</i> to <i>index</i> + <i>length</i> - 1 was greater than <i>value</i> .
The bitwise complement of (<i>index</i> + <i>length</i>).	<i>value</i> was not found, and <i>value</i> was greater than all array elements in the range of <i>index</i> to <i>index</i> + <i>length</i> - 1.

1
2 [Note: If *value* is not found, the caller can take the bitwise complement of the return
3 value to determine the index of *array* where *value* would be found in the range of *index*
4 to *index* + *length* - 1 if *array* is already sorted.]
5
6

7 Description

8 *value* is compared to each element of *array* using *comparer* until an element with a
9 value greater than or equal to *value* is found. If *comparer* is null, the
10 System.IComparable interface of the element being compared - or of *value* if the
11 element being compared does not implement the interface -- is used. If *value* does not
12 implement the System.IComparable interface and is compared to an element that does
13 not implement the System.IComparable interface, a
14 System.InvalidOperationException exception is thrown. If *array* is not already
15 sorted, correct results are not guaranteed.

16
17 [Note: A null reference can be compared with any type; therefore, comparisons with a
18 null reference do not generate exceptions.]
19
20

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.ArgumentOutOfRangeException	<i>index</i> is less than <i>array</i> .GetLowerBound(0). -or- <i>length</i> is less than zero.
System.ArgumentException	<i>index</i> + <i>length</i> is greater than <i>array</i> .GetLowerBound(0) + <i>array</i> .Length.

	-or- <i>array</i> .UpperBound == System.Int32.MaxValue.
System.InvalidOperationException	<i>comparer</i> is null, and both <i>value</i> and at least one element of <i>array</i> do not implement the System.IComparable interface.

1

2 Example

3 This example demonstrates the System.Array.BinarySearch method.

4

5 [C#]

```

6 using System;
7 class BinarySearchExample {
8     public static void Main() {
9         int[] intAry = { 0, 2, 4, 6, 8 };
10        Console.WriteLine( "The indices and elements of the array are: ");
11        for ( int i = 0; i < intAry.Length; i++ )
12            Console.WriteLine( "[{0}]: {1, -5}", i, intAry[i]);
13        Console.WriteLine();
14        SearchFor( intAry, 3 );
15        SearchFor( intAry, 6 );
16        SearchFor( intAry, 9 );
17    }
18    public static void SearchFor( Array ar, Object value ) {
19        int i = Array.BinarySearch( ar, 0, ar.Length, value, null );
20        Console.WriteLine();
21        if ( i > 0 ) {
22            Console.WriteLine( "The object searched for, {0}, was found ", value );
23            Console.WriteLine( "at index {1}.", value, i );
24        }
25        else if ( ~i == ar.Length ) {
26            Console.WriteLine( "The object searched for, {0}, was ", value );
27            Console.WriteLine( "not found,\nand no object in the array had " );
28            Console.WriteLine( "greater value. " );
29        }
30        else {
31            Console.WriteLine( "The object searched for, {0}, was ", value );
32            Console.WriteLine( "not found.\nThe next larger object is at " );
33            Console.WriteLine( "index {0}.", ~i );
34        }
35    }
36 }

```

37

38 The output is

39

40 The indices and elements of the array are:

```
1
2
3 [0]:0 [1]:2 [2]:4 [3]:6 [4]:8
4
5
6 The object searched for, 3, was not found.
7
8
9 The next larger object is at index 2.
10
11
12 The object searched for, 6, was found at index 3.
13
14
15 The object searched for, 9, was not found,
16
17
18 and no object in the array had greater value.
19
20
```

1 Array.BinarySearch(System.Array, 2 System.Object, 3 System.Collections.IComparer) Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 BinarySearch(class System.Array  
6 array, object value, class System.Collections.IComparer comparer)  
  
7 [C#]  
8 public static int BinarySearch(Array array, object value, IComparer  
9 comparer)
```

10 Summary

11 Searches the specified one-dimensional `System.Array` for the specified value, using the
12 specified `System.Collections.IComparer` implementation.

13 Parameters

Parameter	Description
<i>array</i>	A <code>System.Array</code> to search.
<i>value</i>	A <code>System.Object</code> for which to search.
<i>comparer</i>	The <code>System.Collections.IComparer</code> implementation to use when comparing elements. Specify a null reference to use the <code>System.IComparable</code> implementation of each element.

14

15 Return Value

16 A `System.Int32` with one of the following values based on the result of the search
17 operation.

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.
The bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found, and at least one array element was greater than <i>value</i> .
The bitwise complement of $(array.GetLowerBound(0) + array.Length)$.	<i>value</i> was not found, and <i>value</i> was greater than all array elements.

1
2 [Note: If *value* is not found, the caller can take the bitwise complement of the return
3 value to determine the index where *value* would be found in *array* if it is already sorted.]
4
5

6 Description

7 This version of `System.Array.BinarySearch` is equivalent to
8 `System.Array.BinarySearch(array, array.GetLowerBound(0), array.Length, value,`
9 `comparer)`.

10
11 *value* is compared to each element of *array* using *comparer* until an element with a
12 value greater than or equal to *value* is found. If *comparer* is null, the
13 `System.IComparable` interface of the element being compared - or of *value* if the
14 element being compared does not implement the interface - is used. If *value* does not
15 implement the `System.IComparable` interface and is compared to an element that does
16 not implement the `System.IComparable` interface, a
17 `System.InvalidOperationException` exception is thrown. If *array* is not already
18 sorted, correct results are not guaranteed.
19

20 [Note: A null reference can be compared with any type; therefore, comparisons with a
21 null reference do not generate exceptions.]
22
23

24 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.InvalidOperationException	<i>comparer</i> is null, and both <i>value</i> and at least one element of <i>array</i> do not implement the <code>System.IComparable</code> interface.

25

26

1 Array.BinarySearch(System.Array, 2 System.Int32, System.Int32, System.Object) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 BinarySearch(class System.Array  
6 array, int32 index, int32 length, object value)  
  
7 [C#]  
8 public static int BinarySearch(Array array, int index, int length, object  
9 value)
```

10 Summary

11 Searches the specified section of the specified one-dimensional System.Array for the
12 specified value.

13 Parameters

Parameter	Description
<i>array</i>	A System.Array to search.
<i>index</i>	A System.Int32 that contains the index at which searching starts.
<i>length</i>	A System.Int32 that contains the number of elements to search, beginning with <i>index</i> .
<i>value</i>	A System.Object for which to search.

14

15 Return Value

16 A System.Int32 with one of the following values based on the result of the search
17 operation.

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.
The bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found, and at least one array element in the range of <i>index</i> to <i>index</i> + <i>length</i> - 1 was greater than <i>value</i> .
The bitwise complement of (<i>index</i> +	<i>value</i> was not found, and <i>value</i> was greater than all

<i>length</i>).	array elements in the range of <i>index</i> to <i>index</i> + <i>length</i> - 1.
------------------	--

1
2 [Note: If *value* is not found, the caller can take the bitwise complement of the return
3 value to determine the index of the array where *value* would be found in the range of
4 *index* to *index* + *length* - 1 if *array* is already sorted.]
5
6

7 **Description**

8 This version of `System.Array.BinarySearch` is equivalent to
9 `System.Array.BinarySearch(array, array.GetLowerBound(0), array.Length, value,`
10 `null)`.

11 *value* is compared to each element of *array* using the `System.IComparable` interface of
12 the element being compared - or of *value* if the element being compared does not
13 implement the interface - until an element with a value greater than or equal to *value* is
14 found. If *value* does not implement the `System.IComparable` interface and is compared
15 to an element that does not implement the `System.IComparable` interface, a
16 `System.InvalidOperationException` exception is thrown. If *array* is not already
17 sorted, correct results are not guaranteed.
18

19
20 [Note: A null reference can be compared with any type; therefore, comparisons with a
21 null reference do not generate exceptions.]
22
23

24 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.ArgumentOutOfRangeException	<i>index</i> < <i>array</i> .GetLowerBound(0). -or- <i>length</i> < 0.
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> (i.e. <i>index</i> + <i>length</i> > <i>array</i> .GetLowerBound(0) + <i>array</i> .Length).

	-or- <i>array</i> .UpperBound == System.Int32.MaxValue.
System.InvalidOperationException	Either <i>value</i> or at least one element of <i>array</i> does not implement the System.IComparable interface.

1

2

1 `Array.BinarySearch(System.Array,` 2 `System.Object)` Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 BinarySearch(class System.Array  
5 array, object value)  
  
6 [C#]  
7 public static int BinarySearch(Array array, object value)
```

8 **Summary**

9 Searches the specified one-dimensional `System.Array` for the specified object.

10 **Parameters**

Parameter	Description
<i>array</i>	A <code>System.Array</code> to search for an object.
<i>value</i>	A <code>System.Object</code> for which to search.

11

12 **Return Value**

13 A `System.Int32` with one of the following values based on the result of the search
14 operation.

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.
The bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found and the value of at least one element of <i>array</i> was greater than <i>value</i> .
The bitwise complement of $(array.GetLowerBound(0) + array.Length)$.	<i>value</i> was not found, and <i>value</i> was greater than the value of all array elements.

15

16 [Note: If *value* is not found, the caller can take the bitwise complement of the return
17 value to determine the index where *value* would be found in *array* if it is sorted already.]
18
19

1 **Description**

2 This version of `System.Array.BinarySearch` is equivalent to
3 `System.Array.BinarySearch(array, array.GetLowerBound(0), array.Length, value,`
4 `null)`.

5
6 *value* is compared to each element of *array* using the `System.IComparable` interface of
7 the element being compared - or of *value* if the element being compared does not
8 implement the interface - until an element with a value greater than or equal to *value* is
9 found. If *value* does not implement the `System.IComparable` interface and is compared
10 to an element that does not implement the `System.IComparable` interface, a
11 `System.InvalidOperationException` exception is thrown. If *array* is not already
12 sorted, correct results are not guaranteed.

13
14 [Note: A null reference can be compared with any type; therefore, comparisons with a
15 null reference do not generate exceptions.]
16
17

18 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.InvalidOperationException	Both <i>value</i> and at least one element of <i>array</i> do not implement the <code>System.IComparable</code> interface.

19

20

1 Array.BinarySearch<T>(T[], T) Method

```
2 [ILAsm]  
3 .method public hidebysig static int32 BinarySearch<T>(!!0[] array, !!0  
4 value)  
5 [C#]  
6 public static int BinarySearch<T>(T[] array, T value)
```

7 Summary

8 Searches an entire one-dimensional sorted array for a specific element, using the
9 System.IComparable<T> or System.IComparable interface implemented by each
10 element of the array and by the specified object.

11 Parameters

Parameter	Description
<i>array</i>	The one-dimensional array to search.
<i>value</i>	The object for which to search.

12

13 Return Value

14 One of the following values based on the result of the search operation:

Return Value	Description
A non-negative index of <i>value</i> in the array.	<i>value</i> was found.
A negative value, which is the bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found and the value of at least one element of array was greater than <i>value</i> .
A negative value, which is the bitwise complement of one more than the index of the final element.	<i>value</i> was not found, and <i>value</i> was greater than the value of all array elements.

15

16 Description

17 Either *value* or every element of *array* must implement the System.IComparable<T> or
18 System.IComparable interface, which is used for comparisons. The elements of *array*

1 must already be sorted in increasing value according to the sort order defined by the
2 `System.IComparable<T>` or `System.IComparable` implementation; otherwise, the
3 behavior is unspecified
4
5 Duplicate elements are allowed. If the array contains more than one element equal to
6 *value*, the method returns the index of only one of the occurrences, but not necessarily
7 the first one.
8
9 [*Note:* `null` can always be compared with any other reference type; therefore,
10 comparisons with `null` do not generate an exception.]
11
12

13 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is <code>null</code> .
System.InvalidOperationException	Neither <i>value</i> nor the elements of the array implement the <code>System.IComparable<T></code> or <code>System.IComparable</code> interfaces.

14

15

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

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

10 Summary

11 Searches an entire one-dimensional sorted array for a value using the specified
12 System.Collections.Generic.IComparer<T> interface.

13 Parameters

Parameter	Description
<i>array</i>	The one-dimensional array to search.
<i>value</i>	The object for which to search.
<i>comparer</i>	The implementation to use when comparing elements. -or- null to use the System.IComparable<T> or System.IComparable implementation of each element.

14

15 Return Value

16 One of the following values based on the result of the search operation:

Return Value	Description
A non-negative index of <i>value</i> in the array.	<i>value</i> was found.
A negative value, which is the bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found and the value of at least one element of array was greater than <i>value</i> .

A negative value, which is the bitwise complement of one more than the index of the final element.

value was not found, and *value* was greater than the value of all array elements.

1

2 Description

3 The comparer customizes how the elements are compared.

4

5 The elements of *array* must already be sorted in increasing value according to the sort
6 order defined by *comparer*; otherwise, the behavior is unspecified

7

8 If *comparer* is not `null`, the elements of *array* are compared to the specified value using
9 the specified `System.Collections.Generic.IComparer` implementation.

10

11 If *comparer* is `null`, the default comparer is used.

12

13 Duplicate elements are allowed. If the array contains more than one element equal to
14 *value*, the method returns the index of only one of the occurrences, but not necessarily
15 the first one.

16

17 [*Note:* `null` can always be compared with any other reference type; therefore,
18 comparisons with `null` do not generate an exception.]

19

20

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is <code>null</code> .
System.InvalidOperationException	<i>comparer</i> is <code>null</code> , and neither <i>value</i> nor the elements of the array implement the <code>System.IComparable<T></code> or <code>System.IComparable</code> interface.

22

23

1 Array.BinarySearch<T>(T[], System.Int32, 2 System.Int32, T) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 BinarySearch<T>(!!0 array, int32  
5 index, int32 length, !!0 value)  
  
6 [C#]  
7 public static int BinarySearch<T>(T[] array, int index, int length, T  
8 value)
```

9 Summary

10 Searches a range of elements in a one-dimensional sorted array for a value, using the
11 System.IComparable interface implemented by each element of the array and by the
12 specified value.

13 Parameters

Parameter	Description
<i>array</i>	The one-dimensional array to search.
<i>index</i>	The starting index of the range to search.
<i>length</i>	The length of the range to search.
<i>value</i>	The object for which to search.

14 15 Return Value

16 One of the following values based on the result of the search operation:

Return Value	Description
A non-negative index of <i>value</i> in the array.	<i>value</i> was found.
A negative value, which is the bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found and the value of at least one element of array was greater than <i>value</i> .
A negative value, which is the bitwise complement of one more than the index of the final element.	<i>value</i> was not found, and <i>value</i> was greater than the value of all array elements.

1

2 **Description**

3 Either *value* or every element of *array* must implement the `System.IComparable`
4 interface, which is used for comparisons. The elements of *array* must already be sorted
5 in increasing value according to the sort order defined by the `System.IComparable<T>`
6 or `System.IComparable` implementation; otherwise, the behavior is unspecified
7

8 Duplicate elements are allowed. If the array contains more than one element equal to
9 *value*, the method returns the index of only one of the occurrences, but not necessarily
10 the first one.

11
12 [*Note*: `null` can always be compared with any other reference type; therefore,
13 comparisons with `null` do not generate an exception.]
14
15

16 **Exceptions**

Exception	Condition
System.ArgumentException	<i>index</i> + <i>length</i> is greater than <i>array.Length</i> .
System.ArgumentNullException	<i>array</i> is <code>null</code> .
System.ArgumentOutOfRangeException	<i>index</i> is less than zero -or- <i>length</i> is less than zero.
System.InvalidOperationException	Neither <i>value</i> nor the elements of the array implement the <code>System.IComparable<T></code> or <code>System.IComparable</code> interface.

17

18

1 Array.BinarySearch<T>(T[], System.Int32, 2 System.Int32, T, 3 System.Collections.Generic.IComparer<T>) 4 Method

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

12 Summary

13 Searches a range of elements in a one-dimensional sorted array for a value, using the
14 specified `System.Collections.Generic.IComparer<T>` interface.

15 Parameters

Parameter	Description
<i>array</i>	The one-dimensional array to search.
<i>index</i>	The starting index of the range to search.
<i>length</i>	The length of the range to search.
<i>value</i>	The object for which to search.
<i>comparer</i>	The implementation to use when comparing elements. -or- null to use the <code>System.IComparable<T></code> or <code>System.IComparable</code> implementation of each element.

16 17 Return Value

18 One of the following values based on the result of the search operation:

Return Value	Description
--------------	-------------

A non-negative index of <i>value</i> in the array.	<i>value</i> was found.
A negative value, which is the bitwise complement of the index of the first element that is larger than <i>value</i> .	<i>value</i> was not found and the value of at least one element of array was greater than <i>value</i> .
A negative value, which is the bitwise complement of one more than the index of the final element.	<i>value</i> was not found, and <i>value</i> was greater than the value of all array elements.

1

2 Description

3 The comparer customizes how the elements are compared.

4

5 The elements of *array* must already be sorted in increasing value according to the sort
6 order defined by *comparer*; otherwise, the behavior is unspecified.

7

8 If *comparer* is not `null`, the elements of *array* are compared to the specified value using
9 the specified `System.Collections.Generic.IComparer<T>` implementation.

10

11 If *comparer* is `null`, the comparison is done using the `System.IComparable<T>` or
12 `System.IComparable` implementation provided by the element itself or by the specified
13 value.

14

15 Duplicate elements are allowed. If the array contains more than one element equal to
16 *value*, the method returns the index of only one of the occurrences, but not necessarily
17 the first one.

18

19 [*Note:* `null` can always be compared with any other reference type; therefore,
20 comparisons with `null` do not generate an exception.]

21

22

23 Exceptions

Exception	Condition
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in array.
System.ArgumentNullException	<i>array</i> is <code>null</code> .
System.ArgumentOutOfRangeException	<i>index</i> is less than zero -or-

	<i>length</i> is less than zero.
System.InvalidOperationException	<i>comparer</i> is null, and neither <i>value</i> nor the elements of the array implement the System.IComparable<T> or System.IComparable interface.

1

2

1 `Array.Clear(System.Array, System.Int32,` 2 `System.Int32) Method`

```
3 [ILAsm]  
4 .method public hidebysig static void Clear(class System.Array array, int32  
5 index, int32 length)  
  
6 [C#]  
7 public static void Clear(Array array, int index, int length)
```

8 Summary

9 Sets the specified range of elements in the specified `System.Array` to zero, false, or to a
10 null reference, depending on the element type.

11 Parameters

Parameter	Description
<i>array</i>	The <code>System.Array</code> to clear.
<i>index</i>	A <code>System.Int32</code> that contains the index at which clearing starts.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to clear, beginning with <i>index</i> .

12 13 Description

14 Reference-type elements will be set to null. Value-type elements will be set to zero,
15 except for `System.Boolean` elements, which will be set to false.

16 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>index</i> < <i>array</i> .GetLowerBound(0). <i>length</i> < 0. <i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> (i.e. <i>index</i> + <i>length</i> > <i>array</i> .GetLowerBound(0) + <i>array</i> .Length).

1

2

1 `Array.Clone()` Method

```
2 [ILAsm]  
3 .method public hidebysig virtual object Clone()  
4 [C#]  
5 public virtual object Clone()
```

6 **Summary**

7 Returns a `System.Object` that is a copy of the current instance.

8 **Return Value**

9 A `System.Object` that is a copy of the current instance.

10 **Description**

11 [*Note:* This method is implemented to support the `System.ICloneable` interface.]
12
13

14 **Behaviors**

15 Each of the elements of the current instance is copied to the clone. If the elements are
16 reference types, the references are copied. If the elements are value-types, the values
17 are copied. The clone is of the same type as the current instance.

18

19 **Default**

20 As described above.

21

22 **How and When to Override**

23 Override this method to return a clone of an array.

24

25 **Usage**

26 Use this method to obtain the clone of an array.

27

28 **Example**

1 This example demonstrates the System.Array.Clone method.

2

3 [C#]

```
4 using System;
5 public class ArrayCloneExample {
6     public static void Main() {
7         int[] intAryOrig = { 3, 4, 5 };
8         //must explicitly convert clones object into an array
9         int[] intAryClone = (int[]) intAryOrig.Clone();
10        Console.Write( "The elements of the first array are: " );
11        foreach( int i in intAryOrig )
12            Console.Write( "{0,3}", i );
13        Console.WriteLine();
14        Console.Write( "The elements of the cloned array are: " );
15        foreach( int i in intAryClone )
16            Console.Write( "{0,3}", i );
17        Console.WriteLine();
18        //Clear the values of the original array.
19        Array.Clear( intAryOrig, 0, 3 );
20        Console.WriteLine( "After clearing the first array," );
21        Console.Write( "The elements of the first array are: " );
22        foreach( int i in intAryOrig )
23            Console.Write( "{0,3}", i );
24        Console.WriteLine();
25        Console.Write( "The elements of the cloned array are: " );
26        foreach( int i in intAryClone )
27            Console.Write( "{0,3}", i );
28    }
29 }
```

30
31 The output is

32

33 The elements of the first array are: 3 4 5

34

35

36 The elements of the cloned array are: 3 4 5

37

38

39 After clearing the first array,

40

41

42 The elements of the first array are: 0 0 0

43

44

45 The elements of the cloned array are: 3 4 5

46

47

1 `Array.ConvertAll<T,U>(T[],` 2 `System.Converter<T,U>) Method`

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

8 **Summary**

9 Converts an array of one type to an array of another type.

10 **Parameters**

Parameter	Description
<i>array</i>	The one-dimensional array to convert.
<i>converter</i>	A <code>System.Converter<T,U></code> that converts each element from one type to another type.

11

12 **Return Value**

13 A new array of the target type containing the converted elements from *array*.

14 **Description**

15 The `System.Converter<T,U>` is a delegate that converts an array element to the target
16 type. The elements of *array* are individually passed to this converter, and the converted
17 elements are saved in the new array. The source array remains unchanged.

18 **Exceptions**

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

19

20

Array.Copy(System.Array, System.Array, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Copy(class System.Array sourceArray,
class System.Array destinationArray, int32 length)

[C#]
public static void Copy(Array sourceArray, Array destinationArray, int
length)
```

Summary

Copies the specified number of elements from the specified source array to the specified destination array.

Parameters

Parameter	Description
<i>sourceArray</i>	A <code>System.Array</code> that contains the data to copy.
<i>destinationArray</i>	A <code>System.Array</code> that receives the data.
<i>length</i>	A <code>System.Int32</code> designating the number of elements to copy, starting with the first element and proceeding in order.

Description

This version of `System.Array.Copy` is equivalent to `System.Array.Copy (sourceArray, sourceArray.GetLowerBound(0), destinationArray, destinationArray.GetLowerBound(0), length)`.

If *sourceArray* and *destinationArray* are of different types, `System.Array.Copy` performs widening conversions on the elements of *sourceArray* as necessary before storing the information in *destinationArray*. Value types will be boxed when being converted to a `System.Object`. If the necessary conversion is a narrowing conversion, a `System.ArrayTypeMismatchException` exception is thrown. [*Note:* For information regarding valid conversions performed by this method, see `System.Convert`.]

If an exception is thrown while copying, the state of *destinationArray* is undefined.

If *sourceArray* and *destinationArray* are the same array, `System.Array.Copy` copies the source elements safely to their destination, as if the copy were done through an intermediate array.

1 Exceptions

Exception	Condition
System.ArgumentNullException	<i>sourceArray</i> or <i>destinationArray</i> is null.
System.RankException	<i>sourceArray</i> and <i>destinationArray</i> have different ranks.
System.ArrayTypeMismatchException	<p>The elements in both arrays are built-in types, and converting from the type of the elements of <i>sourceArray</i> into the type of the elements in <i>destinationArray</i> requires a narrowing conversion.</p> <p>-or-</p> <p>Both arrays are built-in types, and one array is a value-type array and the other an array of interface type not implemented by that value-type.</p> <p>-or-</p> <p>Both arrays are user-defined value types and are not of the same type.</p>
System.InvalidCastException	At least one of the elements in <i>sourceArray</i> is not assignment-compatible with the type of <i>destinationArray</i> .
System.ArgumentOutOfRangeException	$length < 0$.
System.ArgumentException	$length > sourceArray.Length$. -or- $length > destinationArray.Length$.

2

3 Example

4 This example demonstrates the `System.Array.Copy` method.

5

6 [C#]

```

1  using System;
2  public class ArrayCopyExample {
3      public static void Main() {
4          int[] intAryOrig = new int[3];
5          double[] dAryCopy = new double[3];
6          for ( int i = 0; i < intAryOrig.Length; i++ )
7              intAryOrig[i] = i+3;
8          //copy the first 2 elements of the source into the destination
9          Array.Copy( intAryOrig, dAryCopy, 2);
10         Console.Write( "The elements of the first array are: " );
11         for ( int i = 0; i < intAryOrig.Length; i++ )
12             Console.Write( "{0,3}", intAryOrig[i] );
13         Console.WriteLine();
14         Console.Write( "The elements of the copied array are: " );
15         for ( int i = 0; i < dAryCopy.Length; i++ )
16             Console.Write( "{0,3}", dAryCopy[i] );
17     }
18 }
19

```

20 The output is

21
22 The elements of the first array are: 3 4 5

23
24
25 The elements of the copied array are: 3 4 0

26

27

Array.Copy(System.Array, System.Int32, System.Array, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Copy(class System.Array sourceArray,
int32 sourceIndex, class System.Array destinationArray, int32
destinationIndex, int32 length)

[C#]
public static void Copy(Array sourceArray, int sourceIndex, Array
destinationArray, int destinationIndex, int length)
```

Summary

Copies the specified number of elements from a source array starting at the specified source index to a destination array starting at the specified destination index.

Parameters

Parameter	Description
<i>sourceArray</i>	The <code>System.Array</code> that contains the data to copy.
<i>sourceIndex</i>	A <code>System.Int32</code> that contains the index in <i>sourceArray</i> from which copying begins.
<i>destinationArray</i>	The <code>System.Array</code> that receives the data.
<i>destinationIndex</i>	A <code>System.Int32</code> that contains the index in <i>destinationArray</i> at which storing begins.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to copy.

Description

If *sourceArray* and *destinationArray* are of different types, `System.Array.Copy` performs widening conversions on the elements of *sourceArray* as necessary before storing the information in *destinationArray*. Value types will be boxed when being converted to a `System.Object`. If the necessary conversion is a narrowing conversion, a `System.ArrayTypeMismatchException` exception is thrown. [Note: For information regarding valid conversions performed by this method, see `System.Convert`.]

If an exception is thrown while copying, the state of *destinationArray* is undefined.

1
 2 If *sourceArray* and *destinationArray* are the same array, `System.Array.Copy` copies the
 3 source elements safely to their destination as if the copy were done through an
 4 intermediate array.

5 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>sourceArray</i> or <i>destinationArray</i> is <code>null</code> .
System.RankException	<i>sourceArray</i> and <i>destinationArray</i> have different ranks.
System.ArrayTypeMismatchException	<p>The elements in both arrays are built-in types, and converting from the type of the elements of <i>sourceArray</i> into the type of the elements in <i>destinationArray</i> requires a narrowing conversion.</p> <p>-or-</p> <p>Both arrays are built-in types, and one array is a value-type array and the other an array of interface type not implemented by that value-type.</p> <p>-or-</p> <p>Both arrays are user-defined value types and are not of the same type.</p>
System.InvalidCastException	At least one element in <i>sourceArray</i> is assignment-incompatible with the type of <i>destinationArray</i> .
System.ArgumentOutOfRangeException	<p>$sourceIndex < sourceArray.GetLowerBound(0)$.</p> <p>-or-</p> <p>$destinationIndex < destinationArray.GetLowerBound(0)$.</p> <p>-or-</p>

	<i>length</i> < 0.
System.ArgumentException	<i>(sourceIndex + length) ></i> <i>(sourceArray.GetLowerBound(0) +</i> <i>sourceArray.Length).</i> <i>(destinationIndex + length) ></i> (<i>destinationArray.GetLowerBound(0) +</i> <i>destinationArray.Length).</i>

1

2 **Example**

3 This example demonstrates the System.Array.Copy method.

4

5 [C#]

```

6 using System;
7 class ArrayCopyExample {
8     public static void Main() {
9         int[] intAry = { 0, 10, 20, 30, 40, 50 };
10        Console.Write( "The elements of the array are: " );
11        foreach ( int i in intAry )
12            Console.Write( "{0,3}", i );
13        Console.WriteLine();
14        Array.Copy( intAry, 2, intAry, 0, 4 );
15        Console.WriteLine( "After copying elements 2 through 5 into elements 0
16 through 4" );
17        Console.Write( "The elements of the array are: " );
18        foreach ( int i in intAry )
19            Console.Write( "{0,3}", i );
20        Console.WriteLine();
21    }
22 }

```

23

24 The output is

25

26 The elements of the array are: 0 10 20 30 40 50

27

28

29 After copying elements 2 through 5 into elements 0 through 4

30

31

32 The elements of the array are: 20 30 40 50 40 50

33

34

1 `Array.CopyTo(System.Array, System.Int32)` 2 Method

```
3 [ILAsm]  
4 .method public hidebysig virtual void CopyTo(class System.Array array,  
5 int32 index)  
  
6 [C#]  
7 public virtual void CopyTo(Array array, int index)
```

8 Summary

9 Copies all the elements of the current zero-based instance to the specified one-
10 dimensional array starting at the specified subscript in the destination array.

11 Parameters

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> that is the destination of the elements copied from the current instance.
<i>index</i>	A <code>System.Int32</code> that contains the index in <i>array</i> at which copying begins.

12 13 Description

14 *index* is the array index in the destination array at which copying begins.
15
16 [Note: This method is implemented to support the `System.Collections.ICollection`
17 interface. If implementing `System.Collections.ICollection` is not explicitly required,
18 use `System.Array.Copy` to avoid an extra indirection.
19
20 If this method throws an exception while copying, the state of *array* is undefined.
21
22]

23 Behaviors

24 As described above.

25 26 Default

27 As described above.

28

1 **How and When to Override**

2 Override this method to copy elements of the current instance to a specified array.

3

4 **Usage**

5 Use this method to copy elements of the current instance to a specified array.

6

7 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	The current instance has more than one dimension.
System.ArgumentOutOfRangeException	$index < array.GetLowerBound(0)$.
System.ArgumentException	<i>array</i> has more than one dimension. -or- $(index + \text{Length of the current instance}) > (array.GetLowerBound(0) + array.Length)$. -or- The number of elements in the current instance is greater than the available space from <i>index</i> to the end of <i>array</i> .
System.ArrayTypeMismatchException	The element type of the current instance is not assignment-compatible with the element type of <i>array</i> .

8

9 **Example**

10 The following example shows how to copy the elements of one `System.Array` into
11 another.

12 [C#]
13

```

1  using System;
2
3  public class ArrayCopyToExample
4  {
5      public static void Main()
6      {
7          Array aryOne = Array.CreateInstance(typeof(Object), 3);
8          aryOne.SetValue("one", 0);
9          aryOne.SetValue("two", 1);
10         aryOne.SetValue("three", 2);
11
12         Array aryTwo = Array.CreateInstance(typeof(Object), 5);
13         for (int i=0; i < aryTwo.Length; i++)
14             aryTwo.SetValue(i, i);
15
16         Console.WriteLine("The contents of the first array are:");
17         foreach (object o in aryOne)
18             Console.Write("{0} ", o);
19         Console.WriteLine();
20         Console.WriteLine("The original contents of the second array are:");
21         foreach (object o in aryTwo)
22             Console.Write("{0} ", o);
23         Console.WriteLine();
24
25         aryOne.CopyTo(aryTwo, 1);
26
27         Console.WriteLine("The new contents of the second array are:");
28         foreach( object o in aryTwo)
29             Console.Write("{0} ", o);
30     }
31 }
32 The output is
33
34 The contents of the first array are:
35
36 one two three
37
38 The original contents of the second array are:
39
40 0 1 2 3 4
41
42 The new contents of the second array are:
43
44 0 one two three 4
45

```

1 **The following member must be implemented if the ExtendedArray library is present in the**
2 **implementation.**

3 `Array.CreateInstance(System.Type,` 4 `System.Int32[]) Method`

```
5 [ILAsm]  
6 .method public hidebysig static class System.Array CreateInstance(class  
7 System.Type elementType, int32[] lengths)  
  
8 [C#]  
9 public static Array CreateInstance(Type elementType, int[] lengths)
```

10 Summary

11 Creates a zero-based, multidimensional array of the specified `System.Type` and
12 dimension lengths.

13 Parameters

Parameter	Description
<i>elementType</i>	The <code>System.Type</code> of the elements contained in the new <code>System.Array</code> instance.
<i>lengths</i>	A one-dimensional array of <code>System.Int32</code> objects that contains the size of each dimension of the new <code>System.Array</code> instance.

15 Return Value

16 A new zero-based, multidimensional `System.Array` instance of the specified
17 `System.Type` with the specified length for each dimension. The `System.Array.Rank` of
18 the new instance is equal to *lengths.Length*.

19 Description

20 The number of elements in *lengths* is required to equal the number of dimensions in the
21 new `System.Array` instance. Each element of *lengths* specifies the length of the
22 corresponding dimension in the new instance.

23
24 Reference-type elements will be set to `null`. Value-type elements will be set to zero,
25 except for `System.Boolean` elements, which will be set to `false`.

26
27 [Note: Unlike most classes, `System.Array` provides the `System.Array.CreateInstance`
28 method, instead of public constructors, to allow for late bound access.]
29
30

1 Exceptions

Exception	Condition
System.ArgumentNullException	<i>elementType</i> or <i>lengths</i> is null.
System.ArgumentException	<i>elementType</i> is not a valid System.Type. -or- <i>lengths.Length</i> = 0.
System.ArgumentOutOfRangeException	A value in <i>lengths</i> is less than zero.

2

3 Example

4 The following example shows how to create and initialize a multidimensional
5 System.Array.

6

7 [C#]

8

9 using System;

10

11 public class CreateMultiDimArrayExample

12 {

13 public static void Main()

14 {

15 int i, j, k;

16 int[] indexAry = {2, 4, 5};

17 Array ary = Array.CreateInstance(typeof(int), indexAry);

18 for(i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)

19 {

20 for(j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)

21 {

22 for(k = ary.GetLowerBound(2); k <= ary.GetUpperBound(2); k++)

23 {

24 ary.SetValue((100*i + 10*j + k), i, j, k);

25 }

26 }

27 }

28 Console.WriteLine("The elements of the array are:");

29 for(i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)

30 {

31 for(j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)

32 {

33 for(k = ary.GetLowerBound(2); k <= ary.GetUpperBound(2); k++)

34 {

35 Console.Write("{0, 3} ", ary.GetValue(i, j, k));

36 }

37 Console.WriteLine();

```
1         }
2     Console.WriteLine();
3     }
4 }
5 }
```

7 The output is

```
8 The elements of the array are:
9   0  1  2  3  4
10  10 11 12 13 14
11  20 21 22 23 24
12  30 31 32 33 34
13
14 100 101 102 103 104
15 110 111 112 113 114
16 120 121 122 123 124
17 130 131 132 133 134
18
```

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.CreateInstance(System.Type, System.Int32, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig static class System.Array CreateInstance(class
System.Type elementType, int32 length1, int32 length2, int32 length3)

[C#]
public static Array CreateInstance(Type elementType, int length1, int
length2, int length3)
```

Summary

Creates a zero-based, three-dimensional array of the specified `System.Type` and dimension lengths.

Parameters

Parameter	Description
<i>elementType</i>	The <code>System.Type</code> of the elements contained in the new <code>System.Array</code> instance.
<i>length1</i>	A <code>System.Int32</code> that contains the number of elements contained in the first dimension of the new <code>System.Array</code> instance.
<i>length2</i>	A <code>System.Int32</code> that contains the number of elements contained in the second dimension of the new <code>System.Array</code> instance.
<i>length3</i>	A <code>System.Int32</code> that contains the number of elements contained in the third dimension of the new <code>System.Array</code> instance.

Return Value

A new zero-based, three-dimensional `System.Array` instance of *elementType* objects with the size *length1* for the first dimension, *length2* for the second, and *length3* for the third.

Description

1 Reference-type elements will be set to null. Value-type elements will be set to zero,
2 except for System.Boolean elements, which will be set to false.

3
4 [Note: Unlike most classes, System.Array provides the System.Array.CreateInstance
5 method, instead of public constructors, to allow for late bound access.]
6
7

8 Exceptions

Exception	Condition
System.ArgumentNullException	<i>elementType</i> is null.
System.ArgumentException	<i>elementType</i> is not a valid System.Type.
System.ArgumentOutOfRangeException	<i>length1</i> < 0.
	-or-
	<i>length2</i> < 0.
	-or-
	<i>length3</i> < 0.

9

10 Example

11 The following example shows how to create and initialize a three-dimensional
12 System.Array.

```
13 [C#]  
14  
15 using System;  
16  
17 public class Create3DArrayExample  
18 {  
19     public static void Main()  
20     {  
21         int i, j, k;  
22         Array ary = Array.CreateInstance( typeof(int), 2, 4, 3 );  
23         for( i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++ )  
24         {  
25             for( j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++ )  
26             {  
27                 for( k = ary.GetLowerBound(2); k <= ary.GetUpperBound(2); k++ )  
28                 {  
29                     ary.SetValue( (100*i + 10*j + k), i, j, k );  
30                 }  
29             }  
30         }  
31     }  
32 }
```

```

1         }
2     }
3 }
4 Console.WriteLine("The elements of the array are:");
5 for( i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)
6 {
7     for( j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)
8     {
9         for( k = ary.GetLowerBound(2); k <= ary.GetUpperBound(2); k++ )
10            {
11                Console.Write("{0, 3} ", ary.GetValue(i, j, k));
12            }
13            Console.WriteLine();
14        }
15        Console.WriteLine();
16    }
17 }
18 }
19

```

20 The output is

21 The elements of the array are:

```

22     0   1   2
23    10  11  12
24    20  21  22
25    30  31  32
26
27   100 101 102
28   110 111 112
29   120 121 122
30   130 131 132
31
32

```

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.CreateInstance(System.Type, System.Int32, System.Int32) Method

```
[ILAsm]  
.method public hidebysig static class System.Array CreateInstance(class  
System.Type elementType, int32 length1, int32 length2)
```

```
[C#]  
public static Array CreateInstance(Type elementType, int length1, int  
length2)
```

Summary

Creates a zero-based, two-dimensional array of the specified `System.Type` and dimension lengths.

Parameters

Parameter	Description
<i>elementType</i>	The <code>System.Type</code> of the elements contained in the new <code>System.Array</code> instance.
<i>length1</i>	A <code>System.Int32</code> that contains the number of elements contained in the first dimension of the new <code>System.Array</code> instance.
<i>length2</i>	A <code>System.Int32</code> that contains the number of elements contained in the second dimension of the new <code>System.Array</code> instance.

Return Value

A new zero-indexed, two-dimensional `System.Array` instance of *elementType* objects with the size *length1* for the first dimension and *length2* for the second.

Description

Reference-type elements will be set to `null`. Value-type elements will be set to zero, except for `System.Boolean` elements, which will be set to `false`.

[*Note:* Unlike most classes, `System.Array` provides the `System.Array.CreateInstance` method, instead of public constructors, to allow for late bound access.]

1 Exceptions

Exception	Condition
System.ArgumentNullException	<i>elementType</i> is null.
System.ArgumentException	<i>elementType</i> is not a valid <code>System.Type</code> .
System.ArgumentOutOfRangeException	<i>length1</i> < 0. -or- <i>length2</i> < 0.

2

3 Example

4 The following example shows how to create and initialize a two-dimensional
5 `System.Array`.

6

7 [C#]

8

9 `using System;`

10

11 `public class Create2DArrayExample`

12 `{`

13 `public static void Main()`

14 `{`

15 `int i, j;`

16 `Array ary = Array.CreateInstance(typeof(int), 5, 3);`

17 `for(i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)`

18 `{`

19 `for(j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)`

20 `{`

21 `ary.SetValue((10*i + j), i, j);`

22 `}`

23 `}`

24 `Console.WriteLine("The elements of the array are:");`

25 `for(i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)`

26 `{`

27 `for(j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)`

28 `{`

29 `Console.Write("{0, 2} ", ary.GetValue(i, j));`

30 `}`

31 `Console.WriteLine();`

32 `}`

33 `}`

34 `}`

35

36 The output is

```
1 The elements of the array are:  
2   0  1  2  
3  10 11 12  
4  20 21 22  
5  30 31 32  
6  40 41 42  
7  
8
```

1 `Array.CreateInstance(System.Type,` 2 `System.Int32)` Method

```
3 [ILAsm]  
4 .method public hidebysig static class System.Array CreateInstance(class  
5 System.Type elementType, int32 length)  
  
6 [C#]  
7 public static Array CreateInstance(Type elementType, int length)
```

8 Summary

9 Constructs a zero-based, one-dimensional array with the specified number of elements
10 of the specified type.

11 Parameters

Parameter	Description
<i>elementType</i>	The <code>System.Type</code> of the elements contained in the new <code>System.Array</code> instance.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements contained in the new <code>System.Array</code> instance.

12

13 Return Value

14 A zero-based, one-dimensional `System.Array` object containing *length* elements of type
15 *elementType*.

16 Description

17 Reference-type elements will be set to `null`. Value-type elements will be set to zero,
18 except for `System.Boolean` elements, which will be set to `false`.

19

20 [Note: Unlike most classes, `System.Array` provides the `System.Array.CreateInstance`
21 method, instead of public constructors, to allow for late bound access.]

22

23

24 Exceptions

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

System.ArgumentNullException	<i>elementType</i> is null.
System.ArgumentException	<i>elementType</i> is not a valid System.Type.
System.ArgumentOutOfRangeException	<i>length</i> < 0.

1

2 Example

3 The following example shows how to create and initialize a one-dimensional
4 System.Array.

5

6 [C#]

7 using System;

8

9 public class ArrayCreateInstanceExample

10 {

11

12 public static void Main()

13 {

14

15 Array intAry = Array.CreateInstance(typeof(int),5);

16 for (int i=intAry.GetLowerBound(0);i<=intAry.GetUpperBound(0);i++)

17 intAry.SetValue(i*3,i);

18 Console.WriteLine("The values of the array are:");

19 foreach (int i in intAry)

20 Console.Write("{0} ",i);

21

22 }

23

24 }

25

26 The output is

27

28 The values of the array are: 0 3 6 9 12

29

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.CreateInstance(System.Type, System.Int32[], System.Int32[]) Method

```
[ILAsm]
.method public hidebysig static class System.Array CreateInstance(class
System.Type elementType, int32[] lengths, int32[] lowerBounds)

[C#]
public static Array CreateInstance(Type elementType, int[] lengths, int[]
lowerBounds)
```

Summary

Creates a multidimensional array whose element type is the specified `System.Type`, and dimension lengths and lower bounds, as specified.

Parameters

Parameter	Description
<i>elementType</i>	The <code>System.Type</code> of the elements contained in the new <code>System.Array</code> instance.
<i>lengths</i>	A one-dimensional array of <code>System.Int32</code> objects that contains the size of each dimension of the new <code>System.Array</code> instance.
<i>lowerBounds</i>	A one-dimensional array of <code>System.Int32</code> objects that contains the lower bound of each dimension of the new <code>System.Array</code> instance.

Return Value

A new multidimensional `System.Array` whose element type is the specified `System.Type` and with the specified length and lower bound for each dimension.

Description

The *lengths* and *lowerBounds* are required to have the same number of elements. The number of elements in *lengths* equals the number of dimensions in the new `System.Array` instance

Each element of *lengths* specifies the length of the corresponding dimension in the new `System.Array` instance.

Each element of *lowerBounds* specifies the lower bound of the corresponding dimension in the new `System.Array` instance.

1
 2 Reference-type elements will be set to null. Value-type elements will be set to zero,
 3 except for System.Boolean elements, which will be set to false.
 4
 5 [Note: Unlike most classes, System.Array provides the System.Array.CreateInstance
 6 method, instead of public constructors, to allow for late bound access.]
 7
 8

9 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>elementType</i> , <i>lengths</i> , or <i>lowerBounds</i> is null.
System.ArgumentException	<i>elementType</i> is not a valid System.Type. -or- <i>lengths.Length</i> = 0. -or- <i>lengths</i> and <i>lowerBounds</i> do not contain the same number of elements.
System.ArgumentOutOfRangeException	A value in <i>lengths</i> is less than zero.

10
 11 **Example**

12 The following example shows how to create and initialize a multidimensional
 13 System.Array with specified low bounds.

14 [C#]
 15

```

16
17 using System;
18
19 public class MultiDimNonZeroBoundExample
20 {
21     public static void Main()
22     {
23         int i, j, k;
24         int[] indexAry = {4, 2, 3};
25         int[] lowboundAry = {3, 2, 1};
26         Array ary = Array.CreateInstance( typeof(int), indexAry, lowboundAry );
27         for( i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++ )
28         {
29             for( j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++ )
  
```

```

1      {
2          for( k = ary.GetLowerBound(2); k <= ary.GetUpperBound(2); k++ )
3              {
4                  ary.SetValue( (100*i + 10*j + k), i, j, k );
5              }
6          }
7      }
8      Console.WriteLine("The elements of the array are:");
9      for( i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)
10     {
11         for( j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)
12             {
13                 for( k = ary.GetLowerBound(2); k <= ary.GetUpperBound(2); k++ )
14                     {
15                         Console.Write("{0, 3} ", ary.GetValue(i, j, k));
16                     }
17                 Console.WriteLine();
18             }
19         Console.WriteLine();
20     }
21 }
22 }
23

```

24 The output is

```

25 The elements of the array are:
26 321 322 323
27 331 332 333
28
29 421 422 423
30 431 432 433
31
32 521 522 523
33 531 532 533
34
35 621 622 623
36 631 632 633
37

```

1 `Array.Exists<T>(T[], System.Predicate<T>)`

2 Method

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

8 Summary

9 Determines whether the specified array contains any element that matches the
10 conditions defined by the specified predicate.

11 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>match</i>	The predicate that defines the conditions of the elements to search for.

12

13 Return Value

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

16 Description

17 The predicate returns `true` if the object passed to it matches the delegate. Each element
18 of *array* is passed to the predicate in turn, and processing is stopped when the predicate
19 returns `true`.

20 Exceptions

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

21

22

1 `Array.Find<T>(T[], System.Predicate<T>)` 2 Method

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

8 Summary

9 Searches for an element that matches the predicate, and returns the first occurrence
10 within the entire array.

11 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>match</i>	The predicate that defines the conditions of the element to search for.

12

13 Return Value

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

16 Description

17 The elements of *array* are individually passed to the predicate, moving forward in the
18 array, starting with the first element and ending with the last element. Processing is
19 stopped when the predicate returns `true`.

20 Exceptions

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

21

22

1 `Array.FindAll<T>(T[], System.Predicate<T>)`

2 Method

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

8 Summary

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

10 Parameters

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

11

12 Return Value

13 An array containing all the elements that match the conditions defined by the specified
14 predicate, if found; otherwise, an empty array.

15 Description

16 The elements of *array* are individually passed to the predicate, and those elements for
17 which the predicate returns `true`, are saved in the returned array.

18 Exceptions

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

19

20

1 `Array.FindIndex<T>(T[],` 2 `System.Predicate<T>) Method`

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

8 Summary

9 Searches for an element that matches the predicate, and returns the zero-based index
10 of the first occurrence within the entire array.

11 Parameters

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

12

13 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 elements of *array* are individually passed to the predicate. The array is searched
18 forward starting at the first element and ending at the last element. Processing is
19 stopped when the predicate returns `true`.

20 Exceptions

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

21

22

1 `Array.FindIndex<T>(T[], System.Int32,` 2 `System.Predicate<T>) Method`

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

9 Summary

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

13 Parameters

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

14 15 Return Value

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

18 Description

19 The elements of *array* are individually passed to the predicate. The array is searched
20 forward starting at the specified index and ending at the last element. Processing is
21 stopped when the predicate returns `true`.

22 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>array</i> or <i>match</i> is null.
<code>System.ArgumentOutOfRangeException</code>	<i>startIndex</i> is less than zero or greater than

	<code>array.Length.</code>
--	----------------------------

1

2

1 Array.FindIndex<T>(T[], System.Int32, 2 System.Int32, System.Predicate<T>) Method

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

9 Summary

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

13 Parameters

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

14

15 Return Value

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

18 Description

19 The elements of *array* are individually passed to the predicate. The array is searched
20 forward starting at the specified index and going for *count* elements. Processing is
21 stopped when the predicate returns `true`.

22 Exceptions

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

System.ArgumentNullException	<i>array</i> or <i>match</i> is null.
System.ArgumentOutOfRangeException	<p><i>startIndex</i> is less than zero.</p> <p>-or-</p> <p><i>count</i> is less than zero.</p> <p>-or-</p> <p><i>startIndex</i> + <i>count</i> is greater than <i>array.Length</i>.</p>

1

2

1 `Array.FindLast<T>(T[],` 2 `System.Predicate<T>) Method`

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

8 Summary

9 Searches for an element that matches the predicate, and returns the last occurrence
10 within the entire array.

11 Parameters

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

12

13 Return Value

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

16 Description

17 The elements of *array* are individually passed to the predicate, moving backward in the
18 array, starting with the last element and ending with the first element. Processing is
19 stopped when a match is found.

20 Exceptions

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

21

22

1 `Array.FindLastIndex<T>(T[],` 2 `System.Predicate<T>)` Method

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

8 Summary

9 Searches for an element that matches the predicate, and returns the zero-based index
10 of the last occurrence within the entire array.

11 Parameters

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

12

13 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 elements of *array* are individually passed to the predicate. The array is searched
18 backwards starting at the last element and ending at the first element. Processing is
19 stopped when the predicate returns `true`.

20 Exceptions

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

21

22

1 `Array.FindLastIndex<T>(T[], System.Int32,` 2 `System.Predicate<T>) Method`

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

9 Summary

10 Searches for an element that matches the predicate, and returns the zero-based index
11 of the last occurrence within the range of elements in the array that extends from the
12 specified index to the last element.

13 Parameters

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

14 15 Return Value

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

18 Description

19 The elements of *array* are individually passed to the predicate. The array is searched
20 backward starting at the specified index and ending at the first element. Processing is
21 stopped when the predicate returns `true`.

22 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>array</i> or <i>match</i> is null.
<code>System.ArgumentOutOfRangeException</code>	<i>startIndex</i> is less than zero or greater than

	<code>array.Length.</code>
--	----------------------------

1

2

Array.FindLastIndex<T>(T[], System.Int32, System.Int32, System.Predicate<T>) Method

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

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

Summary

Searches for an element that matches the predicate, and returns the zero-based index of the last occurrence within the range of elements in the array that ends at the specified index and contains the specified number of elements.

Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>startIndex</i>	The zero-based starting index of the backward search.
<i>count</i>	The number of consecutive elements to search.
<i>match</i>	The predicate that specifies the elements 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 elements of *array* are individually passed to the predicate. The array is searched backward starting at the specified index and going for *count* elements. Processing is stopped when the predicate returns `true`.

Exceptions

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

System.ArgumentNullException	<i>array</i> or <i>match</i> is null.
System.ArgumentOutOfRangeException	<p><i>startIndex</i> is less than zero or greater than <i>array.Length</i>.</p> <p>-or-</p> <p><i>count</i> is less than zero.</p> <p>-or-</p> <p><i>count</i> is greater than <i>startIndex + 1</i>.</p>

1

2

1 `Array.ForEach<T>(T[], System.Action<T>)`

2 Method

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

8 Summary

9 Performs the specified action on each element of the specified array.

10 Parameters

Parameter	Description
<i>array</i>	The array on whose elements the action is to be performed.
<i>action</i>	The action to perform on each element of <i>array</i> .

11

12 Description

13 The elements of *array* are individually passed to the action. The elements of the current
14 array are individually passed to the action delegate, sequentially, in index order, and on
15 the same thread as that used to call `ForEach`. Execution stops if the action throws an
16 exception.

17 Exceptions

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

18

19

1 Array.GetEnumerator() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual class System.Collections.IEnumerator  
4 GetEnumerator()  
  
5 [C#]  
6 public virtual IEnumerator GetEnumerator()
```

7 Summary

8 Returns a System.Collections.IEnumerator for the current instance.

9 Return Value

10 A System.Collections.IEnumerator for the current instance.

11 Description

12 A System.Collections.IEnumerator grants read-access to the elements of a
13 System.Array.

14
15 *[Note: This method is implemented to support the System.Collections.IEnumerator*
16 *interface. For more information regarding the use of an enumerator, see*
17 *System.Collections.IEnumerator.]*
18
19

20 Behaviors

21 Enumerators can be used to read the data in the collection, but they cannot be used to
22 modify the underlying collection.
23

24 Initially, the enumerator is positioned before the first element of the current instance.
25 System.Collections.IEnumerator.Reset returns the enumerator to this position.
26 Therefore, after an enumerator is created or after a
27 System.Collections.IEnumerator.Reset,
28 System.Collections.IEnumerator.MoveNext is required to be called to advance the
29 enumerator to the first element of the collection before reading the value of
30 System.Collections.IEnumerator.Current.
31

32 System.Collections.IEnumerator.Current returns the same object until either
33 System.Collections.IEnumerator.MoveNext or
34 System.Collections.IEnumerator.Reset is called.
35 System.Collections.IEnumerator.MoveNext sets
36 System.Collections.IEnumerator.Current to the next element.
37

38 If System.Collections.IEnumerator.MoveNext passes the end of the collection, the
39 enumerator is positioned after the last element in the collection and
40 System.Collections.IEnumerator.MoveNext returns false. When the enumerator is at
41 this position, subsequent calls to System.Collections.IEnumerator.MoveNext also

1 return false. If the last call to `System.Collections.IEnumerator.MoveNext` returned
2 false, `System.Collections.IEnumerator.Current` is unspecified. To set
3 `System.Collections.IEnumerator.Current` to the first element of the collection again,
4 you can call `System.Collections.IEnumerator.Reset` followed by
5 `System.Collections.IEnumerator.MoveNext`.

6
7 An enumerator remains valid as long as the collection remains unchanged. If changes
8 are made to the collection, such as adding, modifying, or deleting elements, the
9 enumerator is irrecoverably invalidated and its behavior is undefined.

10
11 The enumerator does not have exclusive access to the collection; therefore,
12 enumerating through a collection is intrinsically not a thread safe procedure. To
13 guarantee thread safety during enumeration, you can lock the collection during the
14 entire enumeration. To allow the collection to be accessed by multiple threads for
15 reading and writing, you must implement your own synchronization.

16 Default

17 Multidimensional arrays will be processed in Row-major form.

18
19 [*Note:* For some multidimensional `System.Array` objects, it can be desirable for an
20 enumerator to process them in Column-major form.]

23 How and When to Override

24 Override this method to provide read-access to the current instance.

26 Usage

27 Use this method to iterate over the elements of the current instance.

29 Example

30 This example demonstrates the `System.Array.GetEnumerator` method.

31 [C#]

```
32 using System;
33 using System.Collections;
34 public class ArrayGetEnumerator {
35     public static void Main() {
36         string[,] strAry = {{"1","one"}, {"2", "two"}, {"3", "three"}};
37         Console.Write( "The elements of the array are: " );
38         IEnumerator sEnum = strAry.GetEnumerator();
39         while ( sEnum.MoveNext() )
40             Console.Write( " {0}", sEnum.Current );
41     }
```

```
1     }
2   }
3
4   The output is
5
6   The elements of the array are: 1 one 2 two 3 three
7
8
```

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 **Array.GetLength(System.Int32) Method**

```
4 [ILAsm]  
5 .method public hidebysig int32 GetLength(int32 dimension)  
6 [C#]  
7 public int GetLength(int dimension)
```

8 **Summary**

9 Gets the number of elements in the specified dimension of the array.

10 **Parameters**

Parameter	Description
<i>dimension</i>	The zero-based dimension of the array whose length is to be determined.

12 **Return Value**

13 The number of elements in the specified dimension of the array.

14 **Exceptions**

Exception	Condition
System.IndexOutOfRangeException	<i>dimension</i> is less than zero. -or- <i>dimension</i> is equal to or greater than <code>System.Array.Rank</code> .

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 **Array.GetLowerBound(System.Int32) Method**

```
4 [ILAsm]  
5 .method public hidebysig instance int32 GetLowerBound(int32 dimension)  
  
6 [C#]  
7 public int GetLowerBound(int dimension)
```

8 **Summary**

9 Returns the lower bound of the specified dimension in the current instance.

10 **Parameters**

Parameter	Description
<i>dimension</i>	A System.Int32 that contains the zero-based dimension of the current instance whose lower bound is to be determined.

11 12 **Return Value**

13 A System.Int32 that contains the lower bound of the specified dimension in the current
14 instance.

15 **Description**

16 [Note: For example, System.Array.GetLowerBound (0) returns the lower bound of the
17 first dimension of the current instance, and
18 System.Array.GetLowerBound(System.Array.Rank - 1) returns the lower bound of the
19 last dimension of the current instance.]
20
21

22 **Exceptions**

Exception	Condition
System.IndexOutOfRangeException	<i>dimension</i> < 0.
	-or- <i>dimension</i> is equal to or greater than the System.Array.Rank property of the current

	instance.
--	-----------

1

2

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 **Array.GetUpperBound(System.Int32) Method**

```
4 [ILAsm]  
5 .method public hidebysig instance int32 GetUpperBound(int32 dimension)  
  
6 [C#]  
7 public int GetUpperBound(int dimension)
```

8 **Summary**

9 Returns the upper bound of the specified dimension in the current instance.

10 **Parameters**

Parameter	Description
<i>dimension</i>	A System.Int32 that contains the zero-based dimension of the current instance whose upper bound is to be determined.

12 **Return Value**

13 A System.Int32 that contains the upper bound of the specified dimension in the current
14 instance.

15 **Description**

16 [Note: For example, System.Array.GetUpperBound (0) returns the upper bound of the
17 first dimension of the current instance, and
18 System.Array.GetUpperBound(System.Array.Rank - 1) returns the upper bound of the
19 last dimension of the current instance.]
20
21

22 **Exceptions**

Exception	Condition
System.IndexOutOfRangeException	<i>dimension</i> < 0.
	-or- <i>dimension</i> is equal to or greater than the System.Array.Rank property of the current

	instance.
--	-----------

1

2

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.GetValue(System.Int32[]) Method

```
[ILAsm]
.method public hidebysig instance object GetValue(int32[] indices)

[C#]
public object GetValue(int[] indices)
```

Summary

Gets the value at the specified position in the current multidimensional instance.

Parameters

Parameter	Description
<i>indices</i>	A one-dimensional array of <i>System.Int32</i> objects that contains the indices that specify the position of the element in the current instance whose value to get.

Return Value

A *System.Object* that contains the value at the specified position in the current instance.

Description

The number of elements in *indices* is required to be equal to the number of dimensions in the current instance. All elements in *indices* collectively specify the position of the desired element in the current instance.

[*Note:* Use the *System.Array.GetLowerBound* and *System.Array.GetUpperBound* methods to determine whether any of the values in *indices* are out of bounds.]

Exceptions

Exception	Condition
System.ArgumentNullException	<i>indices</i> is null.
System.ArgumentException	The number of dimensions in the current instance is not equal to the number of elements in <i>indices</i> .

System.IndexOutOfRangeException

At least one element in *indices* is outside the range of valid indices for the corresponding dimension of the current instance.

1

2

1 `Array.GetValue(System.Int32)` Method

```
2 [ILAsm]  
3 .method public hidebysig instance object GetValue(int32 index)  
4 [C#]  
5 public object GetValue(int index)
```

6 **Summary**

7 Gets the value at the specified position in the current one-dimensional instance.

8 **Parameters**

Parameter	Description
<i>index</i>	A <code>System.Int32</code> that contains the position of the value to get from the current instance.

9

10 **Return Value**

11 A `System.Object` that contains the value at the specified position in the current
12 instance.

13 **Description**

14 [Note: Use the `System.Array.GetLowerBound` and `System.Array.GetUpperBound`
15 methods to determine whether *index* is out of bounds.]
16
17

18 **Exceptions**

Exception	Condition
System.ArgumentException	The current instance has more than one dimension.
System.IndexOutOfRangeException	<i>index</i> is outside the range of valid indices for the current instance.

19

20 **Example**

21 This example demonstrates the `System.Array.GetValue` method.

22

23 [C#]

```
1 using System;
2 public class ArrayGetValueExample {
3     public static void Main() {
4         String[] strAry = { "one", "two", "three", "four", "five" };
5         Console.Write( "The elements of the array are: " );
6         for( int i = 0; i < strAry.Length; i++ )
7             Console.Write( " '{0}' ", strAry.GetValue( i ) );
8     }
9 }
```

10
11 The output is

12
13 The elements of the array are: 'one' 'two' 'three' 'four' 'five'

14

1 **The following member must be implemented if the ExtendedArray library is present in the**
2 **implementation.**

3 `Array.GetValue(System.Int32, System.Int32)` 4 Method

```
5 [ILAsm]  
6 .method public hidebysig instance object GetValue(int32 index1, int32  
7 index2)  
8 [C#]  
9 public object GetValue(int index1, int index2)
```

10 Summary

11 Gets the value at the specified position in the current two-dimensional instance.

12 Parameters

Parameter	Description
<i>index1</i>	A <code>System.Int32</code> that contains the first-dimension index of the element in the current instance to get.
<i>index2</i>	A <code>System.Int32</code> that contains the second-dimension index of the element in the current instance to get.

14 Return Value

15 A `System.Object` that contains the value at the specified position in the current
16 instance.

17 Description

18 [*Note:* Use the `System.Array.GetLowerBound` and `System.Array.GetUpperBound`
19 methods to determine whether any of the indices are out of bounds.]
20
21

22 Exceptions

Exception	Condition
System.ArgumentException	The current instance does not have exactly two dimensions.

System.IndexOutOfRangeException

At least one of *index1* or *index2* is outside the range of valid indexes for the corresponding dimension of the current instance.

1

2

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.GetValue(System.Int32, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig instance object GetValue(int32 index1, int32
index2, int32 index3)

[C#]
public object GetValue(int index1, int index2, int index3)
```

Summary

Gets the value at the specified position in the current three-dimensional instance.

Parameters

Parameter	Description
<i>index1</i>	A <code>System.Int32</code> that contains the first-dimension index of the element in the current instance to get.
<i>index2</i>	A <code>System.Int32</code> that contains the second-dimension index of the element in the current instance to get.
<i>index3</i>	A <code>System.Int32</code> that contains the third-dimension index of the element in the current instance to get.

Return Value

A `System.Object` that contains the value at the specified position in the current instance.

Description

[*Note:* Use the `System.Array.GetLowerBound` and `System.Array.GetUpperBound` methods to determine whether any of the indices are out of bounds.]

Exceptions

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

System.ArgumentException	The current instance does not have exactly three dimensions.
System.IndexOutOfRangeException	At least one of <i>index1</i> or <i>index2</i> or <i>index3</i> is outside the range of valid indexes for the corresponding dimension of the current instance.

1

2

Array.IndexOf(System.Array, System.Object, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig static int32 IndexOf(class System.Array array,
object value, int32 startIndex, int32 count)

[C#]
public static int IndexOf(Array array, object value, int startIndex, int
count)
```

Summary

Searches the specified one-dimensional `System.Array`, returning the index of the first occurrence of the specified `System.Object` in the specified range.

Parameters

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to search.
<i>value</i>	A <code>System.Object</code> to locate in <i>array</i> .
<i>startIndex</i>	A <code>System.Int32</code> that contains the index at which searching starts.
<i>count</i>	A <code>System.Int32</code> that contains the number of elements to search, beginning with <i>startIndex</i> .

Return Value

A `System.Int32` containing the index of the first occurrence of *value* in *array*, within the range *startIndex* through *startIndex* + *count* - 1, if found; otherwise, *array*.GetLowerBound(0) - 1. [Note: For a vector, if *value* is not found, the return value will be -1. This provides the caller with a standard code for the failed search.]

Description

The elements are compared using `System.Object.Equals`.

Exceptions

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

System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<p><i>startIndex</i> is less than <i>array.GetLowerBound(0)</i>.</p> <p>-or-</p> <p><i>count</i> is less than zero.</p> <p>-or-</p> <p><i>startIndex + count</i> is greater than <i>array.GetLowerBound(0) + array.Length</i>.</p>
System.RankException	<i>array</i> has more than one dimension.

1

2

1 Array.IndexOf(System.Array, System.Object, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 IndexOf(class System.Array array,  
5 object value, int32 startIndex)  
  
6 [C#]  
7 public static int IndexOf(Array array, object value, int startIndex)
```

8 Summary

9 Searches the specified one-dimensional System.Array, returning the index of the first
10 occurrence of the specified System.Object between the specified index and the last
11 element.

12 Parameters

Parameter	Description
<i>array</i>	A one-dimensional System.Array to search.
<i>value</i>	A System.Object to locate in <i>array</i> .
<i>startIndex</i>	A System.Int32 that contains the index at which searching starts.

13

14 Return Value

15 A System.Int32 containing the index of the first occurrence of *value* in *array*, within the
16 range *startIndex* through the last element of *array*, if found; otherwise,
17 *array.GetLowerBound(0) - 1*. [Note: For a vector, if *value* is not found, the return value
18 will be -1. This provides the caller with a standard code for the failed search.]
19
20

21 Description

22 This version of System.Array.IndexOf is equivalent to System.Array.IndexOf (*array*,
23 *value*, *startIndex*, (*array.Length - startIndex + array.GetLowerBound(0)*)).
24

25 The elements are compared using System.Object.Equals.

26 Exceptions

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

System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>startIndex</i> is less than <i>array</i> .GetLowerBound(0) or greater than <i>array</i> .GetLowerBound(0) + <i>array</i> .Length.
System.RankException	<i>array</i> has more than one dimension.

1

2

1 `Array.IndexOf(System.Array, System.Object)` 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 IndexOf(class System.Array array,  
5 object value)  
6 [C#]  
7 public static int IndexOf(Array array, object value)
```

8 Summary

9 Searches the specified one-dimensional `System.Array`, returning the index of the first
10 occurrence of the specified `System.Object`.

11 Parameters

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to search.
<i>value</i>	A <code>System.Object</code> to locate in <i>array</i> .

12 13 Return Value

14 A `System.Int32` containing the index of the first occurrence of *value* in *array*, if found;
15 otherwise, `array.GetLowerBound(0) - 1`. [Note: For a vector, if *value* is not found, the
16 return value will be -1. This provides the caller with a standard code for a failed search.]
17
18

19 Description

20 This version of `System.Array.IndexOf` is equivalent to `System.Array.IndexOf(array,
21 value, array.GetLowerBound(0), array.Length)`.
22

23 The elements are compared using `System.Object.Equals`.

24 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.

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

Example

The following example demonstrates the `System.Array.IndexOf` method.

[C#]

```
using System;
public class ArrayIndexOfExample {
    public static void Main() {
        int[] intAry = { 0, 1, 2, 0, 1 };
        Console.Write( "The values of the array are: " );
        foreach( int i in intAry )
            Console.Write( "{0,5}", i );
        Console.WriteLine();
        int j = Array.IndexOf( intAry, 1 );
        Console.WriteLine( "The first occurrence of 1 is at index {0}", j );
    }
}
```

The output is

The values of the array are: 0 1 2 0 1

The first occurrence of 1 is at index 1

1 `Array.IndexOf<T>(T[], T, System.Int32, System.Int32)` Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 IndexOf<T>(!!0[] array, !!0 value,  
5 int32 startIndex, int32 count)  
  
6 [C#]  
7 public static int IndexOf<T>(T[] array, T value, int startIndex, int  
8 count)
```

9 Summary

10 Searches for the specified value and returns the index of the first occurrence within the
11 range of elements in the array starting at the specified index and continuing for, at
12 most, the specified number of elements.

13 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>value</i>	The value to locate.
<i>startIndex</i>	The zero-based starting index of the search.
<i>count</i>	The number of consecutive elements to search.

14 Return Value

16 The zero-based index of the first occurrence of *value* within the range of elements in
17 *array* that starts at *startIndex* and contains the number of elements specified in *count*, if
18 found; otherwise, -1.

19 Description

20 The elements are compared using `System.Object.Equals`. The array is searched
21 forward starting at *startIndex* and ending at *startIndex* + *count* - 1. Processing is
22 stopped when the predicate returns `true`.

23 Exceptions

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

System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>startIndex</i> is less than zero. -or- <i>count</i> is less than zero. -or- <i>startIndex</i> + <i>count</i> is greater than <code>System.Array.Length</code> .

1

2

1 `Array.IndexOf<T>(T[], T, System.Int32)`

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 IndexOf<T>(!!0[] array, !!0 value,  
5 int32 startIndex)  
  
6 [C#]  
7 public static int IndexOf<T>(T[] array, T value, int startIndex)
```

8 Summary

9 Searches the specified array, returning the index of the first occurrence in the specified
10 array starting at the specified index and including the last element.

11 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>value</i>	The value to locate.
<i>startIndex</i>	The zero-based starting index of the search.

12 Return Value

14 The zero-based index of the first occurrence of *value* within the range of elements in
15 *array* that extends from *startIndex* to the last element, if found; otherwise, -1. If
16 *startIndex* is equal to the length of the array, -1 is returned.

17 Description

18 The elements are compared using `System.Object.Equals`. The array is searched
19 forward starting at *startIndex* and ending at the last element. Processing is stopped
20 when the predicate returns `true`.

21 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>array</i> is null.
<code>System.ArgumentOutOfRangeException</code>	<i>startIndex</i> is less than zero or greater than

	<code>array.Length.</code>
--	----------------------------

1

2

1 Array.IndexOf<T>(T[], T) Method

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

6 Summary

7 Searches the specified array, returning the index of the first occurrence of the specified
8 value.

9 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>value</i>	The value to locate.

10

11 Return Value

12 The zero-based index of the first occurrence of *value* in *array*, if found; otherwise, - 1.

13 Description

14 The elements are compared using `System.Object.Equals`. The array is searched
15 forward starting at the first element and ending at the last element. Processing is
16 stopped when the predicate returns `true`.

17 Exceptions

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

18

19

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 **Array.Initialize() Method**

```
4 [ILAsm]  
5 .method public hidebysig instance void Initialize()  
  
6 [C#]  
7 public void Initialize()
```

8 **Summary**

9 Initializes every element of the current instance of value-type objects by calling the
10 default constructor of that value type.

11 **Description**

12 This method cannot be used on reference-type arrays.

13
14 If the current instance is not a value-type `System.Array` or if the value type does not
15 have a default constructor, the current instance is not modified.

16
17 The current instance can have any lower bound and any number of dimensions.

18
19 [*Note:* This method can be used only on value types that have constructors.]
20
21

22

1 Array.LastIndexOf(System.Array, 2 System.Object, System.Int32, System.Int32) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 LastIndexOf(class System.Array  
6 array, object value, int32 startIndex, int32 count)  
  
7 [C#]  
8 public static int LastIndexOf(Array array, object value, int startIndex,  
9 int count)
```

10 Summary

11 Searches the specified one-dimensional `System.Array`, returning the index of the last
12 occurrence of the specified `System.Object` in the specified range.

13 Parameters

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to search.
<i>value</i>	A <code>System.Object</code> to locate in <i>array</i> .
<i>startIndex</i>	A <code>System.Int32</code> that contains the index at which searching starts.
<i>count</i>	A <code>System.Int32</code> that contains the number of elements to search, beginning with <i>startIndex</i> .

14 15 Return Value

16 A `System.Int32` containing the index of the last occurrence of *value* in *array*, within the
17 range *startIndex* through *startIndex* - *count* + 1, if found; otherwise,
18 *array*.GetLowerBound(0) - 1. [Note: For a vector, if *value* is not found, the return value
19 will be -1. This provides the caller with a standard code for the failed search.]
20
21

22 Description

23 The elements are compared using `System.Object.Equals`.

24 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<p><i>startIndex</i> is outside the range of valid indices for <i>array</i>.</p> <p>-or-</p> <p><i>count</i> < 0.</p> <p>-or-</p> <p><i>count</i> is greater than <i>startIndex</i> + 1.</p>
System.RankException	<i>array</i> has more than one dimension.

1

2

1 `Array.LastIndexOf(System.Array,` 2 `System.Object, System.Int32) Method`

```
3 [ILAsm]  
4 .method public hidebysig static int32 LastIndexOf(class System.Array  
5 array, object value, int32 startIndex)  
  
6 [C#]  
7 public static int LastIndexOf(Array array, object value, int startIndex)
```

8 **Summary**

9 Searches the specified one-dimensional `System.Array`, returning the index of the last
10 occurrence of the specified `System.Object` between the specified index and the first
11 element.

12 **Parameters**

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to search.
<i>value</i>	A <code>System.Object</code> to locate in <i>array</i> .
<i>startIndex</i>	A <code>System.Int32</code> that contains the index at which searching starts.

13

14 **Return Value**

15 A `System.Int32` containing the index of the last occurrence of *value* in the range
16 *startIndex* through the lower bound of *array*, if found; otherwise,
17 *array.GetLowerBound(0) - 1*. [Note: For a vector, if *value* is not found, the return value
18 will be -1. This provides the caller with a standard code for the failed search.]
19
20

21 **Description**

22 This version of `System.Array.LastIndexOf` is equivalent to
23 `System.Array.LastIndexOf(array, value, startIndex, startIndex + 1 -`
24 `array.GetLowerBound(0))`.
25

26 The elements are compared using `System.Object.Equals`.

27 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>startIndex</i> is outside the range of valid indices for <i>array</i> .
System.RankException	<i>array</i> has more than one dimension.

1

2

1 `Array.LastIndexOf(System.Array,` 2 `System.Object)` Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 LastIndexOf(class System.Array  
5 array, object value)  
  
6 [C#]  
7 public static int LastIndexOf(Array array, object value)
```

8 Summary

9 Searches the specified one-dimensional `System.Array`, returning the index of the last
10 occurrence of the specified `System.Object`.

11 Parameters

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to search.
<i>value</i>	A <code>System.Object</code> to locate in <i>array</i> .

12

13 Return Value

14 A `System.Int32` containing the index of the last occurrence in *array* of *value*, if found;
15 otherwise, `array.GetLowerBound(0) - 1`. [Note: For a vector, if *value* is not found, the
16 return value will be -1. This provides the caller with a standard code for the failed
17 search.]

18

19

20 Description

21 This version of `System.Array.LastIndexOf` is equivalent to
22 `System.Array.LastIndexOf(array, value, (array.GetLowerBound(0) + array.Length -`
23 `1), array.Length)`.

24

25 The elements are compared using `System.Object.Equals`.

26 Exceptions

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

1

2 Example

3 The following example demonstrates the `System.Array.LastIndexOf` method.

4

5 [C#]

6 `using System;`

7

8 `public class ArrayLastIndexOfExample {`

9

10 `public static void Main() {`11 `int[] intAry = { 0, 1, 2, 0, 1 };`12 `Console.Write("The values of the array are: ");`13 `foreach(int i in intAry)`14 `Console.Write("{0,5}", i);`15 `Console.WriteLine();`16 `int j = Array.LastIndexOf(intAry, 1);`17 `Console.WriteLine("The last occurrence of 1 is at index {0}", j);`18 `}`19 `}`

20 The output is

21

22 The values of the array are: 0 1 2 0 1

23

24

25 The last occurrence of 1 is at index 4

26

27

1 Array.LastIndexOf<T>(T[], T, System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 LastIndexOf<T>(!!0[] array, !!0  
5 value, int32 startIndex, int32 count)  
  
6 [C#]  
7 public static int LastIndexOf<T>(T[] array, T value, int startIndex, int  
8 count)
```

9 Summary

10 Searches for the specified value and returns the index of the last occurrence within the
11 range of elements in the array starting at the specified index and continuing backwards
12 for, at most, the specified number of elements.

13 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>value</i>	The value to locate.
<i>startIndex</i>	The zero-based starting index of the search.
<i>count</i>	The number of consecutive elements to search.

14 15 Return Value

16 The zero-based index of the last occurrence of *value* within the range of elements in
17 *array* that ends at *startIndex* and contains the number of elements specified in *count*, if
18 found; otherwise, -1.

19 Description

20 The elements are compared using `System.Object.Equals`. The array is searched
21 backward starting at *startIndex* and going for *count* elements. Processing is stopped
22 when the predicate returns `true`.

23 Exceptions

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

System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>startIndex</i> is outside the range of valid indices for <i>array</i> . -or- <i>count</i> is less than zero. -or- <i>count</i> is greater than <i>startIndex</i> + 1.

1

2

1 Array.LastIndexOf<T>(T[], T, System.Int32)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 LastIndexOf<T>(!!0[] array, !!0  
5 value, int32 startIndex)  
  
6 [C#]  
7 public static int LastIndexOf<T>(T[] array, T value, int startIndex)
```

8 Summary

9 Searches the specified array backwards, returning the index of the last occurrence of the
10 specified array, starting at the specified index.

11 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>value</i>	The value to locate.
<i>startIndex</i>	The zero-based starting index of the search.

12

13 Return Value

14 The zero-based index of the last occurrence of *value* within the range of elements in
15 *array* that extends from *startIndex* to the first element, if found; otherwise, -1.

16 Description

17 The elements are compared using `System.Object.Equals`. The array is searched
18 backward starting at *startIndex* and ending at the first element. Processing is stopped
19 when the predicate returns `true`.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>startIndex</i> is outside the range of valid indices for <i>array</i> .

1

2

1 Array.LastIndexOf<T>(T[], T) Method

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

7 Summary

8 Searches the specified array, returning the index of the last occurrence of the specified
9 value.

10 Parameters

Parameter	Description
<i>array</i>	The array to search.
<i>value</i>	The value to locate.

11

12 Return Value

13 The zero-based index of the last occurrence of *value* in *array*, if found; otherwise, - 1.

14 Description

15 The elements are compared using `System.Object.Equals`. The array is searched
16 backward starting at the last element and ending at the first element. Processing is
17 stopped when the predicate returns `true`.

18 Exceptions

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

19

20

1 Array.Resize<T>(T[], System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig static void Resize<T>(!!0[]& array, int32  
4 newSize)  
5 [C#]  
6 public static void Resize<T>(ref T[] array, int newSize)
```

7 Summary

8 Changes the size of an array to the specified new size.

9 Parameters

Parameter	Description
<i>array</i>	The array to resize. -or- null to create a new array with the specified size.
<i>newSize</i>	The size of the new array.

10

11 Description

12 If array is null, this method creates a new array with the specified size.

13

14 If array is not null, then if *newSize* is equal to `System.Array.Length` of the old array, this method does nothing. Otherwise, this method allocates a new array with the specified size, copies elements from the old array to the new one, and then assigns the new array reference to the array parameter. If *newSize* is greater than `System.Array.Length` of the old array, a new array is allocated and all the elements are copied from the old array to the new one. If *newSize* is less than `System.Array.Length` of the old array, a new array is allocated and elements are copied from the old array to the new one until the new one is filled; the rest of the elements in the old array are ignored.

23 Exceptions

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

24

25

1 Array.Reverse(System.Array, System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Reverse(class System.Array array,  
5 int32 index, int32 length)  
  
6 [C#]  
7 public static void Reverse(Array array, int index, int length)
```

8 Summary

9 Reverses the sequence of the elements in the specified range of the specified one-
10 dimensional `System.Array`.

11 Parameters

Parameter	Description
<i>array</i>	The one-dimensional <code>System.Array</code> to reverse.
<i>index</i>	A <code>System.Int32</code> that contains the index at which reversing starts.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to reverse.

12

13 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> is multidimensional.
System.ArgumentOutOfRangeException	<i>index</i> < <i>array</i> .GetLowerBound(0). <i>length</i> < 0.
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> (i.e. <i>index</i> + <i>length</i> > <i>array</i> .GetLowerBound(0) + <i>array</i> .Length).

14

15 Example

1 The following example demonstrates the System.Array.Reverse method.

2

3 [C#]

```
4 using System;
5 public class ArrayReverseExample {
6     public static void Main() {
7         string[] strAry = { "one", "two", "three" };
8         Console.Write( "The elements of the array are:");
9         foreach( string str in strAry )
10            Console.Write( " {0}", str );
11        Array.Reverse( strAry );
12        Console.WriteLine();
13        Console.WriteLine( "After reversing the array," );
14        Console.Write( "the elements of the array are:");
15        foreach( string str in strAry )
16            Console.Write( " {0}", str );
17    }
18 }
```

19 The output is

20

21 The elements of the array are: one two three

22

23

24 After reversing the array,

25

26

27 the elements of the array are: three two one

28

29

1 `Array.Reverse(System.Array)` Method

```
2 [ILAsm]  
3 .method public hidebysig static void Reverse(class System.Array array)  
4 [C#]  
5 public static void Reverse(Array array)
```

6 **Summary**

7 Reverses the sequence of the elements in the specified one-dimensional `System.Array`.

8 **Parameters**

Parameter	Description
<code>array</code>	The one-dimensional <code>System.Array</code> to reverse.

9 **Description**

11 This version of `System.Array.Reverse` is equivalent to `System.Array.Reverse(array, array.GetLowerBound(0), array.Length)`.
12

13 **Exceptions**

Exception	Condition
<code>System.ArgumentNullException</code>	<code>array</code> is null.
<code>System.RankException</code>	<code>array</code> has more than one dimension.

14

15

1 Array.SetValue(System.Object, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance void SetValue(object value, int32 index)  
5 [C#]  
6 public void SetValue(object value, int index)
```

7 Summary

8 Sets the value of the element at the specified position in the current one-dimensional
9 instance.

10 Parameters

Parameter	Description
<i>value</i>	A System.Object that contains the new value for the specified element.
<i>index</i>	A System.Int32 that contains the index of the element whose value is to be set.

11

12 Description

13 [Note: Use the System.Array.GetLowerBound and System.Array.GetUpperBound
14 methods to determine whether *index* is out of bounds.

15

16 For more information regarding valid conversions that will be performed by this method,
17 see System.Convert.

18

19]

20 Exceptions

Exception	Condition
System.ArgumentException	The current instance has more than one dimension.
System.IndexOutOfRangeException	<i>index</i> is outside the range of valid indices for the current instance.
System.InvalidCastException	<i>value</i> is not assignment-compatible with the element type of the current instance.

1

2

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.SetValue(System.Object, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig instance void SetValue(object value, int32
index1, int32 index2)

[C#]
public void SetValue(object value, int index1, int index2)
```

Summary

Sets the value of the element at the specified position in the current two-dimensional instance.

Parameters

Parameter	Description
<i>value</i>	A <code>System.Object</code> that contains the new value for the specified element.
<i>index1</i>	A <code>System.Int32</code> that contains the first-dimension index of the element in the current instance to set.
<i>index2</i>	A <code>System.Int32</code> that contains the second-dimension index of the element in the current instance to set.

Description

[*Note:* For more information regarding valid conversions that will be performed by this method, see `System.Convert`.

Use the `System.Array.GetLowerBound` and `System.Array.GetUpperBound` methods to determine whether any of the indices are out of bounds.

]

Exceptions

Exception	Condition
System.ArgumentException	The current instance does not have exactly two dimensions.

System.IndexOutOfRangeException	At least one of <i>index1</i> or <i>index2</i> is outside the range of valid indices for the corresponding dimension of the current instance.
System.InvalidCastException	<i>value</i> is not assignment-compatible with the element type of the current instance.

1

2

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.SetValue(System.Object, System.Int32, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig instance void SetValue(object value, int32
index1, int32 index2, int32 index3)

[C#]
public void SetValue(object value, int index1, int index2, int index3)
```

Summary

Sets the value of the element at the specified position in the current three-dimensional instance.

Parameters

Parameter	Description
<i>value</i>	A <code>System.Object</code> that contains the new value for the specified element.
<i>index1</i>	A <code>System.Int32</code> that contains the first-dimension index of the element in the current instance to set.
<i>index2</i>	A <code>System.Int32</code> that contains the second-dimension index of the element in the current instance to set.
<i>index3</i>	A <code>System.Int32</code> that contains the third-dimension index of the element in the current instance to set.

Description

[*Note:* For more information regarding valid conversions that will be performed by this method, see `System.Convert`.

Use the `System.Array.GetLowerBound` and `System.Array.GetUpperBound` methods to determine whether any of the indices are out of bounds.

]

Exceptions

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

System.ArgumentException	The current instance does not have exactly three dimensions.
System.IndexOutOfRangeException	At least one of <i>index1</i> , <i>index2</i> , or <i>index3</i> is outside the range of valid indices for the corresponding dimension of the current instance.
System.InvalidCastException	<i>value</i> is not assignment-compatible with the element type of the current instance.

1

2

The following member must be implemented if the ExtendedArray library is present in the implementation.

Array.SetValue(System.Object, System.Int32[]) Method

```
[ILAsm]
.method public hidebysig instance void SetValue(object value, int32[]
indices)

[C#]
public void SetValue(object value, int[] indices)
```

Summary

Sets the value of the element at the specified position in the current multidimensional instance.

Parameters

Parameter	Description
<i>value</i>	A <code>System.Object</code> that contains the new value for the specified element.
<i>indices</i>	A one-dimensional array of <code>System.Int32</code> objects that contains the indices that specify the position of the element in the current instance to set.

Description

The number of elements in *indices* is required to be equal to the number of dimensions in the current instance. All elements in *indices* collectively specify the position of the desired element in the current instance.

[*Note:* For more information regarding valid conversions that will be performed by this method, see `System.Convert`.

Use the `System.Array.GetLowerBound` and `System.Array.GetUpperBound` methods to determine whether any of the values in *indices* is out of bounds.

]

Exceptions

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

System.ArgumentException	The number of dimensions in the current instance is not equal to the number of elements in <i>indices</i> .
System.IndexOutOfRangeException	At least one element in <i>indices</i> is outside the range of valid indices for the corresponding dimension of the current instance.
System.InvalidCastException	<i>value</i> is not assignment-compatible with the element type of the current instance.

1

2

1 Array.Sort(System.Array, System.Array, 2 System.Int32, System.Int32, 3 System.Collections.IComparer) Method

```
4 [ILAsm]  
5 .method public hidebysig static void Sort(class System.Array keys, class  
6 System.Array items, int32 index, int32 length, class  
7 System.Collections.IComparer comparer)  
  
8 [C#]  
9 public static void Sort(Array keys, Array items, int index, int length,  
10 IComparer comparer)
```

11 Summary

12 Sorts the specified range of the specified pair of one-dimensional `System.Array` objects
13 (one containing a set of keys and the other containing corresponding items) based on
14 the keys in the first specified `System.Array` using the specified
15 `System.Collections.IComparer` implementation.

16 Parameters

Parameter	Description
<i>keys</i>	A one-dimensional <code>System.Array</code> that contains the keys to sort.
<i>items</i>	A one-dimensional <code>System.Array</code> that contains the items that correspond to each element of <i>keys</i> . Specify a null reference to sort only <i>keys</i> .
<i>index</i>	A <code>System.Int32</code> that contains the index at which sorting starts.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to sort.
<i>comparer</i>	The <code>System.Collections.IComparer</code> implementation to use when comparing elements. Specify a null reference to use the <code>System.IComparable</code> implementation of each element.

17

18 Description

19 Each key in *keys* is required to have a corresponding item in *items*. The sort is
20 performed according to the order of *keys*. After a key is repositioned during the sort, the
21 corresponding item in *items* is similarly repositioned. Only *keys.Length* elements of
22 *items* will be sorted. Therefore, *items* is sorted according to the arrangement of the
23 corresponding keys in *keys*. If the sort is not successfully completed, the results are
24 undefined.

1
 2 If *comparer* is a null reference, each element of *keys* is required to implement the
 3 `System.IComparable` interface to be capable of comparisons with every other element
 4 in *keys*.

5 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>keys</i> is null.
System.RankException	<i>keys</i> has more than one dimension. -or- <i>items</i> is not a null reference and has more than one dimension.
System.ArgumentOutOfRangeException	<i>index</i> < <i>keys</i> .GetLowerBound(0). -or- <i>length</i> < 0.
System.ArgumentException	<i>items</i> is not a null reference, and <i>keys</i> .GetLowerBound(0) does not equal <i>items</i> .GetLowerBound(0). -or- <i>index</i> and <i>length</i> do not specify a valid range in <i>key</i> . -or- <i>items</i> is not a null reference, and <i>index</i> and <i>length</i> do not specify a valid range in <i>items</i> .
System.InvalidOperationException	<i>comparer</i> is null, and one or more elements in <i>keys</i> that are used in a comparison do not implement the <code>System.IComparable</code> interface.

6

7

1 `Array.Sort(System.Array, System.Int32,` 2 `System.Int32, System.Collections.IComparer)` 3 **Method**

```
4 [ILAsm]  
5 .method public hidebysig static void Sort(class System.Array array, int32  
6 index, int32 length, class System.Collections.IComparer comparer)  
  
7 [C#]  
8 public static void Sort(Array array, int index, int length, IComparer  
9 comparer)
```

10 **Summary**

11 Sorts the elements in the specified section of the specified one-dimensional
12 `System.Array` using the specified `System.Collections.IComparer` implementation.

13 **Parameters**

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to sort.
<i>index</i>	A <code>System.Int32</code> that contains the index at which sorting starts.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to sort.
<i>comparer</i>	The <code>System.Collections.IComparer</code> implementation to use when comparing elements. Specify a null reference to use the <code>System.IComparable</code> implementation of each element.

14

15 **Description**

16 This version of `System.Array.Sort` is equivalent to `System.Array.Sort(array, null,`
17 `index, length, comparer)`.

18

19 If *comparer* is a null reference, each element of *array* is required to implement the
20 `System.IComparable` interface to be capable of comparisons with every other element
21 in *array*. If the sort is not successfully completed, the results are unspecified.

22 **Exceptions**

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

System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.ArgumentOutOfRangeException	<i>index</i> < <i>array</i> .GetLowerBound(0). -or- <i>length</i> < 0.
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> .
System.InvalidOperationException	<i>comparer</i> is null, and one or more elements in <i>array</i> that are used in a comparison do not implement the <i>System.IComparable</i> interface.

1

2

Array.Sort(System.Array, System.Array, System.Collections.IComparer) Method

```
[ILAsm]
.method public hidebysig static void Sort(class System.Array keys, class
System.Array items, class System.Collections.IComparer comparer)

[C#]
public static void Sort(Array keys, Array items, IComparer comparer)
```

Summary

Sorts the specified pair of one-dimensional `System.Array` objects (one containing a set of keys and the other containing corresponding items) based on the keys in the first specified `System.Array` using the specified `System.Collections.IComparer` implementation.

Parameters

Parameter	Description
<i>keys</i>	A one-dimensional <code>System.Array</code> that contains the keys to sort.
<i>items</i>	A one-dimensional <code>System.Array</code> that contains the items that correspond to each element in <i>keys</i> . Specify a null reference to sort only <i>keys</i> .
<i>comparer</i>	The <code>System.Collections.IComparer</code> implementation to use when comparing elements. Specify a null reference to use the <code>System.IComparable</code> implementation of each element.

Description

This version of `System.Array.Sort` is equivalent to `System.Array.Sort(keys, items, keys.GetLowerBound(0), keys.Length, comparer)`.

Each key in *keys* is required to have a corresponding item in *items*. The sort is performed according to the order of *keys*. After a key is repositioned during the sort, the corresponding item in *items* is similarly repositioned. Only *keys.Length* elements of *items* are sorted. Therefore, *items* is sorted according to the arrangement of the corresponding keys in *keys*. If the sort is not successfully completed, the results are unspecified.

If *comparer* is a null reference, each element of *keys* is required to implement the `System.IComparable` interface to be capable of comparisons with every other element in *keys*.

Exceptions

Exception	Condition
System.ArgumentNullException	<i>keys</i> is null.
System.RankException	<i>keys</i> has more than one dimension. -or- <i>items</i> is not a null reference and has more than one dimension.
System.ArgumentException	<i>items</i> is not a null reference, and <i>keys</i> .GetLowerBound(0) does not equal <i>items</i> .GetLowerBound(0). -or- <i>items</i> is not a null reference, and <i>keys</i> .Length > <i>items</i> .Length.
System.InvalidOperationException	<i>comparer</i> is a null, and one or more elements in <i>keys</i> that are used in a comparison do not implement the <code>System.IComparable</code> interface.

1

2

1 `Array.Sort(System.Array,` 2 `System.Collections.IComparer)` Method

```
3 [ILAsm]  
4 .method public hidebysig static void Sort(class System.Array array, class  
5 System.Collections.IComparer comparer)  
  
6 [C#]  
7 public static void Sort(Array array, IComparer comparer)
```

8 Summary

9 Sorts the elements in the specified one-dimensional `System.Array` using the specified
10 `System.Collections.IComparer` implementation.

11 Parameters

Parameter	Description
<i>array</i>	The one-dimensional <code>System.Array</code> to sort.
<i>comparer</i>	The <code>System.Collections.IComparer</code> implementation to use when comparing elements. Specify a null reference to use the <code>System.IComparable</code> implementation of each element.

12 13 Description

14 This version of `System.Array.Sort` is equivalent to `System.Array.Sort(array, null,`
15 `array.GetLowerBound(0), array.Length, comparer)`.

16
17 If *comparer* is a null reference, each element of *array* is required to implement the
18 `System.IComparable` interface to be capable of comparisons with every other element
19 in *array*. If the sort is not successfully completed, the results are unspecified.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.InvalidOperationException	<i>comparer</i> is a null reference, and one or more elements in <i>array</i> that are used in a comparison do

not implement the `System.IComparable` interface.

1

2

Array.Sort(System.Array, System.Array, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Sort(class System.Array keys, class
System.Array items, int32 index, int32 length)

[C#]
public static void Sort(Array keys, Array items, int index, int length)
```

Summary

Sorts the specified ranges of the specified pair of one-dimensional `System.Array` objects (one containing a set of keys and the other containing corresponding items) based on the keys in the first specified `System.Array`.

Parameters

Parameter	Description
<i>keys</i>	A one-dimensional <code>System.Array</code> that contains the keys to sort.
<i>items</i>	A one-dimensional <code>System.Array</code> that contains the items that correspond to each element in <i>keys</i> . Specify a null reference to sort only <i>keys</i> .
<i>index</i>	A <code>System.Int32</code> that contains the index at which sort begins.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to sort.

Description

This version of `System.Array.Sort` is equivalent to `System.Array.Sort(keys, items, index, length, null)`.

Each key in *keys* is required to have a corresponding item in *items*. The sort is performed according to the order of *keys*. After a key is repositioned during the sort, the corresponding item in *items* is similarly repositioned. Therefore, *items* is sorted according to the arrangement of the corresponding keys in *keys*. If the sort is not successfully completed, the results are undefined.

Each element of *keys* is required to implement the `System.IComparable` interface to be capable of comparisons with every other element in *keys*.

Exceptions

Exception	Condition
System.ArgumentNullException	<i>keys</i> is null.
System.RankException	<p><i>keys</i> has more than one dimension.</p> <p>-or-</p> <p><i>items</i> is not a null reference and has more than one dimension.</p>
System.ArgumentOutOfRangeException	<p>$index < keys.GetLowerBound(0)$.</p> <p>-or-</p> <p>$length < 0$.</p>
System.ArgumentException	<p><i>items</i> is not a null reference, and $keys.GetLowerBound(0)$ does not equal $items.GetLowerBound(0)$.</p> <p>-or-</p> <p><i>index</i> and <i>length</i> do not specify a valid range in <i>keys</i>.</p> <p>-or-</p> <p><i>items</i> is not a null reference, and <i>index</i> and <i>length</i> do not specify a valid range in <i>items</i>.</p>
System.InvalidOperationException	One or more elements in <i>keys</i> that are used in a comparison do not implement the <code>System.IComparable</code> interface.

1

2

1 Array.Sort(System.Array) Method

```
2 [ILAsm]  
3 .method public hidebysig static void Sort(class System.Array array)  
4 [C#]  
5 public static void Sort(Array array)
```

6 Summary

7 Sorts the elements of the specified one-dimensional `System.Array`.

8 Parameters

Parameter	Description
<code>array</code>	A one-dimensional <code>System.Array</code> to sort.

9

10 Description

11 This version of `System.Array.Sort` is equivalent to `System.Array.Sort(array, null,`
12 `array.GetLowerBound(0), array.Length, null)`.

13

14 Each element of `array` is required to implement the `System.IComparable` interface to be
15 capable of comparisons with every other element in array.

16 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<code>array</code> is null.
<code>System.RankException</code>	<code>array</code> has more than one dimension.
<code>System.InvalidOperationException</code>	One or more elements in <code>array</code> that are used in a comparison do not implement the <code>System.IComparable</code> interface.

17

18 Example

19 This example demonstrates the `System.Array.Sort` method.

20

21 [C#]

```
22 using System;  
23 public class ArraySortExample {
```

```
1     public static void Main() {
2         string[] strAry = { "All's", "well", "that", "ends", "well" };
3         Console.Write( "The original string array is: " );
4         foreach ( String str in strAry )
5             Console.Write( str + " " );
6         Console.WriteLine();
7         Array.Sort( strAry );
8         Console.Write( "The sorted string array is: " );
9         foreach ( string str in strAry )
10            Console.Write( str + " " );
11     }
12 }
```

13 The output is

14
15 The original string array is: All's well that ends well

16
17
18 The sorted string array is: All's ends that well well

19

20

1 `Array.Sort(System.Array, System.Array)`

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static void Sort(class System.Array keys, class  
5 System.Array items)  
  
6 [C#]  
7 public static void Sort(Array keys, Array items)
```

8 Summary

9 Sorts the specified pair of one-dimensional `System.Array` objects (one containing a set
10 of keys and the other containing corresponding items) based on the keys in the first
11 specified `System.Array`.

12 Parameters

Parameter	Description
<i>keys</i>	A one-dimensional <code>System.Array</code> that contains the keys to sort.
<i>items</i>	A one-dimensional <code>System.Array</code> that contains the items that correspond to each of element of <i>keys</i> . Specify a null reference to sort only <i>keys</i> .

13 14 Description

15 This version of `System.Array.Sort` is equivalent to `System.Array.Sort(keys, items,`
16 `keys.GetLowerBound(0), keys.Length, null)`.

17
18 Each key in *keys* is required to have a corresponding item in *items*. The sort is
19 performed according to the order of *keys*. After a key is repositioned during the sort, the
20 corresponding item in *items* is similarly repositioned. Only *keys.Length* elements of
21 *items* are sorted. Therefore, *items* is sorted according to the arrangement of the
22 corresponding keys in *keys*. If the sort is not successfully completed, the results are
23 unspecified.

24
25 Each element of *keys* is required to implement the `System.IComparable` interface to be
26 capable of comparisons with every other element in *keys*.

27 Exceptions

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

System.RankException	<p><i>keys</i> has more than one dimension.</p> <p>-or-</p> <p><i>items</i> is not a null reference and has more than one dimension.</p>
System.ArgumentException	<p><i>items</i> is not a null reference, and <i>keys</i>.GetLowerBound(0) does not equal <i>items</i>.GetLowerBound(0).</p> <p>-or-</p> <p><i>items</i> is not a null reference, and <i>keys</i>.Length > <i>items</i>.Length.</p>
System.InvalidOperationException	<p>One or more elements in <i>keys</i> that are used in a comparison do not implement the System.IComparable interface.</p>

1

2 **Example**

3 This example demonstrates the System.Array.Sort method.

4

5 [C#]

```

6 using System;
7 public class ArraySortExample {
8     public static void Main() {
9         string[] strAry = { "All's", "well", "that", "ends", "well" };
10        int[] intAry = { 3, 4, 0, 1, 2 };
11        Console.Write( "The original string array is: " );
12        foreach ( string str in strAry )
13            Console.Write( str + " " );
14        Console.WriteLine();
15        Console.Write( "The key array is: " );
16        foreach ( int i in intAry )
17            Console.Write( i + " " );
18        Console.WriteLine();
19        Array.Sort( intAry, strAry );
20        Console.Write( "The sorted string array is: " );
21        foreach ( string str in strAry )
22            Console.Write( str + " " );
23    }
24 }

```

25 The output is

26

27 The original string array is: All's well that ends well

28

```
1
2 The key array is: 3 4 0 1 2
3
4
5 The sorted string array is: that ends well All's well
6
7
```

1 `Array.Sort(System.Array, System.Int32, System.Int32)` Method

```
3 [ILAsm]  
4 .method public hidebysig static void Sort(class System.Array array, int32  
5 index, int32 length)  
  
6 [C#]  
7 public static void Sort(Array array, int index, int length)
```

8 Summary

9 Sorts the elements in the specified range of the specified one-dimensional
10 `System.Array`.

11 Parameters

Parameter	Description
<i>array</i>	A one-dimensional <code>System.Array</code> to sort.
<i>index</i>	A <code>System.Int32</code> that contains the index at which sorting starts.
<i>length</i>	A <code>System.Int32</code> that contains the number of elements to sort.

12 13 Description

14 This version of `System.Array.Sort` is equivalent to `System.Array.Sort(array, null,`
15 `index, length, null)`.

16
17 Each element of *array* is required to implement the `System.IComparable` interface to be
18 capable of comparisons with every other element in *array*. If the sort is not successfully
19 completed, the results are unspecified.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.RankException	<i>array</i> has more than one dimension.
System.ArgumentOutOfRangeException	$index < array.GetLowerBound(0)$. -or-

	<i>length</i> < 0.
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> .
System.InvalidOperationException	One or more elements in <i>array</i> that are used in a comparison do not implement the <code>System.IComparable</code> interface.

1

2

1 **Array.Sort<K,V>(K[], V[], System.Int32,**
2 **System.Int32,**
3 **System.Collections.Generic.IComparer<K>)**
4 **Method**

```
5 [ILAsm]  
6 .method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items,  
7 int32 index, int32 length, class  
8 System.Collections.Generic.IComparer`1<!!0> comparer)  
  
9 [C#]  
10 public static void Sort<K,V>(K[] keys, V[] items, int index, int length,  
11 IComparer<K> comparer)
```

12 **Summary**

13 Sorts a range of elements in a pair of arrays based on the keys in the first array using
14 the specified `System.Collections.Generic.IComparer<K>`.

15 **Parameters**

Parameter	Description
<i>keys</i>	The array that contains the keys to sort.
<i>items</i>	The array that contains the items that correspond to each of the keys in <i>keys</i> . -or- null to sort only the <i>keys</i> array.
<i>index</i>	The starting index of the range to sort.
<i>length</i>	The number of elements in the range to sort.
<i>comparer</i>	The <code>System.Collections.Generic.IComparer<K></code> implementation to use when comparing elements. -or- null to use the <code>System.IComparable<K></code> or <code>System.IComparable</code> implementation of each element.

16
17 **Description**

1 If *items* is non-null, each key in *keys* is required to have a corresponding item in *items*.
 2 The sort is performed according to the order of *keys*. After a key is repositioned during
 3 the sort, the corresponding item in *items* is similarly repositioned. Only *keys.Length*
 4 elements of *items* will be sorted. Therefore, *items* is sorted according to the
 5 arrangement of the corresponding keys in *keys*. If the sort is not successfully
 6 completed, the results are undefined.

7
 8 If *comparer* is a null reference, each element of *keys* is required to implement the
 9 `System.IComparable<K>` or `System.IComparable` interface to be capable of
 10 comparisons with every other element in *keys*.

11 **Exceptions**

Exception	Condition
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>keys</i> . -or- <i>items</i> is not null, and <i>index</i> and <i>length</i> do not specify a valid range in <i>items</i> .
System.ArgumentNullException	<i>keys</i> is null.
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>length</i> is less than zero.
System.InvalidOperationException	<i>comparer</i> is null, and one or more elements in <i>keys</i> that are used in a comparison do not implement the <code>System.IComparable<K></code> or <code>System.IComparable</code> interface.

12

13

1 `Array.Sort<K,V>(K[], V[],` 2 `System.Collections.Generic.IComparer<K>)` 3 **Method**

```
4 [ILAsm]  
5 .method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items,  
6 class System.Collections.Generic.IComparer`1<!!0> comparer)  
  
7 [C#]  
8 public static void Sort<K,V>(K[] keys, V[] items, IComparer<K> comparer)
```

9 **Summary**

10 Sorts a pair of arrays based on the keys in the first array, using the specified
11 `System.Collections.Generic.IComparer`.

12 **Parameters**

Parameter	Description
<i>keys</i>	The array that contains the keys to sort.
<i>items</i>	The array that contains the items that correspond to each of the keys in <i>keys</i> . -or- null to sort only the <i>keys</i> array.
<i>comparer</i>	The <code>System.Collections.Generic.IComparer<K></code> implementation to use when comparing elements. -or- null to use the <code>System.IComparable<K></code> or <code>System.IComparable</code> implementation of each element.

13 14 **Description**

15 This version of `System.Array.Sort` is equivalent to `System.Array.Sort<K,V>(keys,`
16 `items, 0, keys.Length, comparer)`.

17
18 If *items* is non-null, each key in *keys* is required to have a corresponding item in *items*.
19 The sort is performed according to the order of *keys*. After a key is repositioned during
20 the sort, the corresponding item in *items* is similarly repositioned. Only *keys.Length*
21 elements of *items* will be sorted. Therefore, *items* is sorted according to the

1 arrangement of the corresponding keys in *keys*. If the sort is not successfully
2 completed, the results are unspecified.
3
4 If *comparer* is a null reference, each element of *keys* is required to implement the
5 `System.IComparable<K>` or `System.IComparable` interface to be capable of
6 comparisons with every other element in *keys*.

7 **Exceptions**

Exception	Condition
System.ArgumentNullException	<i>keys</i> is null.
System.ArgumentException	<i>items</i> is not null, and the length of <i>keys</i> does not match the length of <i>items</i> .
System.InvalidOperationException	<i>comparer</i> is null, and one or more elements in <i>keys</i> that are used in a comparison do not implement the <code>System.IComparable<K></code> or <code>System.IComparable</code> interface.

8

9

1 Array.Sort<K,V>(K[], V[], System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items,  
5 int32 index, int32 length)  
  
6 [C#]  
7 public static void Sort<K,V>(K[] keys, V[] items, int index, int length)
```

8 Summary

9 Sorts a range of elements in a pair of arrays based on the keys in the first array, using
10 the System.IComparable<K> or System.IComparable implementation of each key.

11 Parameters

Parameter	Description
<i>keys</i>	The array that contains the keys to sort.
<i>items</i>	The array that contains the items that correspond to each of the keys in <i>keys</i> . -or- null to sort only the <i>keys</i> array.
<i>index</i>	The starting index of the range to sort.
<i>length</i>	The number of elements in the range to sort.

12

13 Description

14 If *items* is non-null, each key in *keys* is required to have a corresponding item in *items*.
15 When a key is repositioned during the sorting, the corresponding item in *items* is
16 similarly repositioned. Therefore, *items* is sorted according to the arrangement of the
17 corresponding keys in *keys*.

18

19 If the sort is not successfully completed, the results are unspecified.

20

21 Each key within the specified range of elements in *keys* must implement the
22 System.IComparable<K> or System.IComparable interface to be capable of
23 comparisons with every other key.

24

25 This implementation performs an unstable sort; that is, if two elements are equal, their

1 order might not be preserved. In contrast, a stable sort preserves the order of elements
2 that are equal.

3 **Exceptions**

Exception	Condition
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>keys</i> . -or- <i>items</i> is not null, and <i>index</i> and <i>length</i> do not specify a valid range in <i>items</i> .
System.ArgumentNullException	<i>keys</i> is null.
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>length</i> is less than zero.
System.InvalidOperationException	One or more elements in <i>keys</i> that are used in a comparison are the null reference or do not implement the <code>System.IComparable<K></code> or <code>System.IComparable</code> interface.

4

5

1 `Array.Sort<K,V>(K[], V[])` Method

```
2 [ILAsm]  
3 .method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items)  
4 [C#]  
5 public static void Sort<K,V>(K[] keys, V[] items)
```

6 Summary

7 Sorts a pair of arrays based on the keys in the first array using the
8 `System.IComparable` implementation of each key.

9 Parameters

Parameter	Description
<i>keys</i>	The array that contains the keys to sort.
<i>items</i>	The array that contains the items that correspond to each of the keys in <i>keys</i> . -or- null to sort only the <i>keys</i> array.

10

11 Description

12 If *items* is non-null, each key in *keys* is required to have a corresponding item in *items*.
13 When a key is repositioned during the sorting, the corresponding item in *items* is
14 similarly repositioned. Therefore, *items* is sorted according to the arrangement of the
15 corresponding keys in *keys*.

16

17 Each key in *keys* must implement the `System.IComparable<K>` or `System.IComparable`
18 interface to be capable of comparisons with every other key.

19

20 If the sort is not successfully completed, the results are undefined.

21

22 This implementation performs an unstable sort; that is, if two elements are equal, their
23 order might not be preserved. In contrast, a stable sort preserves the order of elements
24 that are equal.

25 Exceptions

Exception	Condition
<code>System.ArgumentException</code>	<i>items</i> is not null, and the length of <i>keys</i> does not

	equal the length of <i>items</i> .
System.ArgumentNullException	<i>keys</i> is null.
System.InvalidOperationException	One or more elements in <i>keys</i> that are used in a comparison are the null reference or do not implement the <code>System.IComparable<K></code> or <code>System.IComparable</code> interface.

1

2

1 `Array.Sort<T>(T[], System.Int32,` 2 `System.Int32,` 3 `System.Collections.Generic.IComparer<T>)` 4 **Method**

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

11 **Summary**

12 Sorts the elements in a range of elements in an array using the specified comparer.

13 **Parameters**

Parameter	Description
<i>array</i>	The array to sort.
<i>index</i>	The starting index of the range to sort.
<i>length</i>	The number of elements in the range to sort.
<i>comparer</i>	The <code>System.Collections.Generic.IComparer<K></code> implementation to use when comparing elements. -or- null to use the <code>System.IComparable<K></code> or <code>System.IComparable</code> implementation of each element.

14 15 **Description**

16 If *comparer* is null, each element within the specified range of elements in *array* must
17 implement the `System.IComparable` interface to be capable of comparisons with every
18 other element in *array*.

19 If the sort is not successfully completed, the results are undefined.

20 This implementation performs an unstable sort; that is, if two elements are equal, their
21
22

1 order might not be preserved. In contrast, a stable sort preserves the order of elements
2 that are equal.

3 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>length</i> is less than zero.
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> .
System.InvalidOperationException	<i>comparer</i> is null, and one or more elements in <i>array</i> that are used in a comparison do not implement the <code>System.IComparable<K></code> or <code>System.IComparable</code> interface.

4

5

1 `Array.Sort<T>(T[],` 2 `System.Collections.Generic.IComparer<T>)` 3 **Method**

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

9 **Summary**

10 Sorts the elements in an array using the specified comparer.

11 **Parameters**

Parameter	Description
<i>array</i>	The array to sort.
<i>comparer</i>	The <code>System.Collections.Generic.IComparer<T></code> implementation to use when comparing elements. -or- null to use the <code>System.IComparable<T></code> or <code>System.IComparable</code> implementation of each element.

12

13 **Description**

14 If *comparer* is null, each element of *array* must implement the `System.IComparable<T>`
15 or `System.IComparable` interface to be capable of comparisons with every other
16 element in *array*.

17

18 If the sort is not successfully completed, the results are undefined.

19

20 This implementation performs an unstable sort; that is, if two elements are equal, their
21 order might not be preserved. In contrast, a stable sort preserves the order of elements
22 that are equal.

23 **Exceptions**

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

System.ArgumentNullException	<i>array</i> is null.
System.InvalidOperationException	<i>comparer</i> is null, and one or more elements in <i>array</i> that are used in a comparison do not implement the <code>System.IComparable<T></code> or <code>System.IComparable</code> interface.

1

2

1 `Array.Sort<T>(T[], System.Comparison<T>)`

2 Method

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

8 Summary

9 Sorts the elements in an array using the specified comparison.

10 Parameters

Parameter	Description
<i>array</i>	The array to sort.
<i>comparison</i>	The <code>System.Comparison<T></code> to use when comparing elements.

11

12 Description

13 If the sort is not successfully completed, the results are undefined.

14

15 This implementation performs an unstable sort; that is, if two elements are equal, their
16 order might not be preserved. In contrast, a stable sort preserves the order of elements
17 that are equal.

18 Exceptions

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

19

20

1 Array.Sort<T>(T[]) Method

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

6 Summary

7 Sorts the elements in an entire array using the `System.IComparable<T>` or
8 `System.IComparable` implementation of each element of that array.

9 Parameters

Parameter	Description
<i>array</i>	The array to sort.

10

11 Description

12 Each element of *array* is required to implement the `System.IComparable<T>` or
13 `System.IComparable` interface to be capable of comparisons with every other element
14 in *array*.

15

16 If the sort is not successfully completed, the results are undefined.

17

18 This implementation performs an unstable sort; that is, if two elements are equal, their
19 order might not be preserved. In contrast, a stable sort preserves the order of elements
20 that are equal.

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>array</i> is null.
System.InvalidOperationException	One or more elements in <i>array</i> that are used in a comparison are the null reference or do not implement the <code>System.IComparable<T></code> or <code>System.IComparable</code> interface.

22

23

1 Array.Sort<T>(T[], System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Sort<T>(!!0[] array, int32 index,  
5 int32 length)  
  
6 [C#]  
7 public static void Sort<T>(T[] array, int index, int length)
```

8 Summary

9 Sorts an array using the System.IComparable<T> or System.IComparable
10 implementation of each element of that array.

11 Parameters

Parameter	Description
<i>array</i>	The array to sort.
<i>index</i>	The starting index of the range to sort.
<i>length</i>	The number of elements in the range to sort.

12 13 Description

14 Each element within the specified range of elements in *array* must implement the
15 System.IComparable<T> or System.IComparable interface to be capable of
16 comparisons with every other element in *array*.

17
18 If the sort is not successfully completed, the results are undefined.

19
20 This implementation performs an unstable sort; that is, if two elements are equal, their
21 order might not be preserved. In contrast, a stable sort preserves the order of elements
22 that are equal.

23 Exceptions

Exception	Condition
System.ArgumentException	<i>index</i> and <i>length</i> do not specify a valid range in <i>array</i> .
System.ArgumentNullException	<i>array</i> is null.

System.ArgumentOutOfRangeException	<i>index</i> is less than zero. -or- <i>length</i> is less than zero.
System.InvalidOperationException	One or more elements in <i>array</i> that are used in a comparison do not implement the <code>System.IComparable<T></code> or <code>System.IComparable</code> interface.

1

2

1
2
3
4
5
6
7
8
9
10
11
12

Array.System.Collections.IList.Add(System.Object) Method

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

Summary

Implemented to support the System.Collections.IList interface. [Note: For more information, see System.Collections.IList.Add.]

1 **Array.System.Collections.IList.Clear()**

2 **Method**

```
3 [ILAsm]  
4 .method private final hidebysig virtual void  
5 System.Collections.IList.Clear()
```

```
6 [C#]  
7 void IList.Clear()
```

8 **Summary**

9 Implemented to support the `System.Collections.IList` interface. [Note: For more
10 information, see `System.Collections.IList.Clear`.]

11

1
2
3
4
5
6
7
8
9
10
11
12

Array.System.Collections.IList.Contains(System.Object) Method

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

Summary

Implemented to support the System.Collections.IList interface. [Note: For more information, see System.Collections.IList.Contains.]

1
2
3
4
5
6
7
8
9
10
11
12

Array.System.Collections.IList.IndexOf(System.Object) Method

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

Summary

Implemented to support the System.Collections.IList interface. [Note: For more information, see System.Collections.IList.IndexOf.]

1
2 **Array.System.Collections.IList.Insert(System.**
3 **Int32, System.Object) Method**

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

9 **Summary**

10 Implemented to support the System.Collections.IList interface. [Note: For more
11 information, see System.Collections.IList.Insert.]

12

1
2
3
4
5
6
7
8
9
10
11
12

Array.System.Collections.IList.Remove(System.Object) Method

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

Summary

Implemented to support the System.Collections.IList interface. [Note: For more information, see System.Collections.IList.Remove.]

1
2
3
4
5
6
7
8
9
10
11
12

Array.System.Collections.IList.RemoveAt(System.Int32) Method

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

Summary

Implemented to support the System.Collections.IList interface. [Note: For more information, see System.Collections.IList.RemoveAt.]

1 Array.TrueForAll<T>(T[], 2 System.Predicate<T>) Method

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

8 Summary

9 Determines whether every element in the array matches the predicate.

10 Parameters

Parameter	Description
<i>array</i>	The array to check against the conditions.
<i>match</i>	The predicate against which the elements are checked..

11

12 Return Value

13 `true`, if every element in *array* matches the specified predicate; otherwise, `false`.

14 Description

15 The predicate returns `true` if the object passed to it matches the delegate. The elements
16 of *array* are individually passed to the predicate, and processing is stopped when the
17 delegate returns `false` for any element.

18 Exceptions

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

19

20

1 Array.IsFixedSize Property

```
2 [ILAsm]  
3 .property public bool IsFixedSize { public hidebysig virtual abstract  
4 specialname bool get_IsFixedSize() }  
5 [C#]  
6 public bool IsFixedSize { get; }
```

7 Summary

8 Implemented to support the `System.Collections.IList` interface. [Note: For more
9 information, see `System.Collections.IList.IsFixedSize`.]

10

1 Array.IsReadOnly Property

```
2 [ILAsm]  
3 .property public bool IsReadOnly { public hidebysig virtual abstract  
4 specialname bool get_IsReadOnly() }  
  
5 [C#]  
6 public bool IsReadOnly { get; }
```

7 Summary

8 Implemented to support the `System.Collections.IList` interface. [Note: For more
9 information, see `System.Collections.IList.IsReadOnly`.]

10

1 Array.IsSynchronized Property

```
2 [ILAsm]  
3 property public bool IsSynchronized { public hidebysig virtual abstract  
4 specialname bool get_IsSynchronized() }
```

```
5 [C#]  
6 public bool IsSynchronized { get; }
```

7 Summary

8 Implemented to support the `System.Collections ICollection` interface. [Note: For
9 more information, see `System.Collections ICollection.IsSynchronized`.]

10

1 Array.Length Property

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

7 Summary

8 Gets the total number of elements in all the dimensions of the current instance.

9 Property Value

10 A `System.Int32` that contains the total number of elements in all the dimensions of the
11 current instance.

12 Description

13 This property is read-only.

14

1 Array.LongLength Property

```
2 [ILAsm]  
3 .property int64 Length { public hidebysig specialname instance int64  
4 get_LongLength() }  
5 [C#]  
6 public long LongLength {get;}
```

7 Summary

8 Gets the total number of elements in all the dimensions of the current instance.

9 Property Value

10 A `System.Int64` value containing the length of the array.

11 Description

12 This property is read-only.

13

1 Array.Rank Property

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

7 Summary

8 Gets the rank (number of dimensions) of the current instance.

9 Property Value

10 A `System.Int32` that contains the rank (number of dimensions) of the current instance.

11 Description

12 This property is read-only.

13

1 Array.SyncRoot Property

```
2 [ILAsm]  
3 .property public object SyncRoot { public hidebysig virtual abstract  
4 specialname object get_SyncRoot() }
```

```
5 [C#]  
6 public object SyncRoot { get; }
```

7 Summary

8 Implemented to support the `System.Collections.ICollection` interface. [Note: For
9 more information, see `System.Collections.ICollection.SyncRoot`.]

10

1 Array.System.Collections.ICollection.Count

2 Property

```
3 [ILAsm]  
4 .property int32 ICollection.Count { public hidebysig virtual abstract  
5 specialname int32 get_ICollection.Count() }
```

```
6 [C#]  
7 int ICollection.Count { get; }
```

8 Summary

9 Implemented to support the System.Collections.ICollection interface. [Note: For
10 more information, see System.Collections.ICollection.Count.]

11

1 Array.System.Collections.IList.Item Property

```
2 [ILAsm]  
3 .property object IList.Item[int32 index] { public hidebysig virtual  
4 abstract specialname object get_IList.Item(int32 index) public hidebysig  
5 virtual abstract specialname void set_IList.Item(int32 index, object  
6 value) }  
  
7 [C#]  
8 public virtual object this[int index] { get; set; }
```

9 Summary

10 Implemented to support the System.Collections.IList interface. [Note: For more
11 information, see System.Collections.IList.Item.]

12