

System.Array Class

```
[ILASM]
.class public abstract serializable Array extends
System.Object implements System.ICloneable,
System.Collections.ICollection,
System.Collections.IEnumerable, System.Collections.IList

[C#]
public abstract class Array: ICloneable, ICollection,
IEnumerable, IList
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- *Version:* 1.0.x.x
- *Attributes:*
 - CLSCompliantAttribute(true)

Implements:

- **System.ICloneable**
- **System.Collections.IList**
- **System.Collections.ICollection**
- **System.Collections.IEnumerable**

Summary

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

Inherits From: System.Object

Library: BCL

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

Description

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

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

26
27
28
29

30
31
32
33
34
35

36
37
38
39
40

41
42
43
44
45
46
47

The *rank* of an array is the number of dimensions in the array. Each dimension has its own set of indices. An array with a rank greater than one can have a different lower bound and a different number of elements for each dimension. Multidimensional arrays (i.e. arrays with a rank greater than one) are processed in row-major order.

The *lower bound* of a dimension is the starting index of that dimension.

The *length* of an array is the total number of elements contained in all of its dimensions.

A *vector* is a one-dimensional array with a *lower bound* of '0'.

If the implementer creates a derived class of **System.Array**, expected **System.Array** behavior cannot be guaranteed. For information on array-like objects with increased functionality, see the **System.Collections.IList** interface. For more information regarding the use of arrays versus the use of collections, see Partition V of the CLI Specification.] Every specific **System.Array** type has three instance methods defined on it. While some programming languages allow direct access to these methods, they are primarily intended to be called by the output of compilers based on language syntax that deals with arrays.

- **Get**: Takes as many **System.Int32** arguments as the array has dimensions and returns the value stored at the given index. It throws a **System.IndexOutOfRangeException** exception for invalid indices.
- **Set**: Takes as many **System.Int32** arguments as the array has dimensions, plus one additional argument (the last argument) which has the same type as an array element. It stores the final value in the specified index of the array. It throws a **System.IndexOutOfRangeException** exception for invalid indices.
- **Address**: Takes as many **System.Int32** arguments as the array has dimensions and returns the address of the element at the given index. It throws a **System.IndexOutOfRangeException** exception for invalid indices.

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14

dimension indexed with 0, 1, 2, and 3.

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

1 Array() Constructor

```
2 [ILASM]  
3 family specialname instance void .ctor()  
4 [C#]  
5 protected Array()
```

6 Summary

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

8

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

```
3 [ILASM]  
4 .method public hidebysig static int32 BinarySearch(class  
5 System.Array 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
10 specified object.

11 Parameters

12
13

Parameter	Description
<i>array</i>	A System.Array to search for an object.
<i>value</i>	A System.Object for which to search, or a null reference. [Note: A null reference will be considered to compare less than any non-null object, or equal to another null reference.]

14
15
16

Return Value

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

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.
The bitwise complement 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 (<i>array</i> .GetLowerBound(0) + <i>array</i> .Length).	<i>value</i> was not found, and <i>value</i> was greater than the value of all array elements.

19
20
21
22

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

23 Description

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

4
5 *value* is compared to each element of *array* using the
6 **System.IComparable** interface of the element being compared - or
7 of *value* if the element being compared does not implement the
8 interface - until an element with a value greater than or equal to *value*
9 is found. If *value* does not implement the **System.IComparable**
10 interface and is compared to an element that does not implement the
11 **System.IComparable** interface, a **System.ArgumentException**
12 exception is thrown. If *array* is not already sorted, correct results are
13 not guaranteed.

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

17 Exceptions

18

19

Exception	Condition
System.ArgumentException	Both <i>value</i> and at least one element of <i>array</i> do not implement the System.IComparable interface. -or- <i>value</i> is not assignment-compatible with at least one element of <i>array</i> . -or- <i>array</i> .UpperBound == System.Int32.MaxValue .
System.ArgumentNullException	<i>array</i> is null .
System.RankException	<i>array</i> has more than one dimension.

20

21

22

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

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

11 Summary

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

14 Parameters

15
16

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, or a null reference. [<i>Note:</i> A null reference will be considered to compare less than any non-null object, or equal to another null reference.]

17
18
19

Return Value

20 A **System.Int32** with one of the following values based on the result
21 of the search operation.

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.
The bitwise complement 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> 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
2
3
4
5

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

6 **Description**

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

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

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

23 **Exceptions**

24
25

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

1
2
3

value is not assignment-compatible with at least one element of *array*.

-or-

array.UpperBound ==
System.Int32.MaxValue.

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

```
4 [ILASM]  
5 .method public hidebysig static int32 BinarySearch(class  
6 System.Array array, object value, class  
7 System.Collections.IComparer comparer)
```

```
8 [C#]  
9 public static int BinarySearch(Array array, object value,  
10 IComparer comparer)
```

11 Summary

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

15 Parameters

16
17

Parameter	Description
<i>array</i>	A System.Array to search.
<i>value</i>	A System.Object for which to search, or a null reference. [Note: A null reference will be considered to compare less than any non-null object, or equal to another null reference.]
<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.

18

19 Return Value

20

21 A **System.Int32** with one of the following values based on the result
22 of the search operation.

Return Value	Description
The index of <i>value</i> in the array.	<i>value</i> was found.
The bitwise complement 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
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

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

Description

This version of **System.Array.BinarySearch** is equivalent to **System.Array.BinarySearch(array, array.GetLowerBound(0), array.Length, value, comparer)**.

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

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

Exceptions

Exception	Condition
System.ArgumentException	<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. -or- <i>comparer</i> is null , and <i>value</i> is not assignment-compatible with at least one element of <i>array</i> . -or- <i>array</i> .UpperBound == System.Int32.MaxValue .
System.ArgumentNullException	<i>array</i> is null .
System.RankException	<i>array</i> has more than one dimension.

25
26
27

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

```
5 [ILASM]  
6 .method public hidebysig static int32 BinarySearch(class  
7 System.Array array, int32 index, int32 length, object  
8 value, class System.Collections.IComparer comparer)
```

```
9 [C#]  
10 public static int BinarySearch(Array array, int index, int  
11 length, object value, IComparer comparer)
```

12 **Summary**

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

16 **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, or a null reference. [Note: A null reference will be considered to compare less than any non-null object, or equal to another null reference.]
<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.

19
20 **Return Value**

21

22 A **System.Int32** with one of the following values based on the result
23 of the search operation.

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

The bitwise complement 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 + length</i> was greater than <i>value</i> .
The bitwise complement of (<i>index + 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 + length</i> .

1
2
3
4
5

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

6 **Description**

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

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

19 **Exceptions**

20
21

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 + length</i> > <i>array</i> .GetLowerBound(0) + <i>array</i> .Length). -or- <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.

-or-

comparer is **null**, and *value* is not of the same type as the elements of *array*.

-or-

array.UpperBound == **System.Int32.MaxValue**.

1
2
3

Example

4
5
6
7

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

8
9
10

[C#]

```
using System;
class BinarySearchExample {
    public static void Main() {
        int[] intAry = { 0, 2, 4, 6, 8 };
        Console.WriteLine("The indices and elements of the
array are: ");
        for (int i = 0; i < intAry.Length; i++)
            Console.WriteLine("[{0}]: {1, -5}", i, intAry[i]);
        Console.WriteLine();
        SearchFor(intAry, 3);
        SearchFor(intAry, 6);
        SearchFor(intAry, 9);
    }
    public static void SearchFor(Array ar, Object value) {
        int i = Array.BinarySearch(ar, 0, ar.Length, value,
null);
        Console.WriteLine();
        if (i > 0) {
            Console.WriteLine("The object searched for, {0}, was
found ", value);
            Console.WriteLine("at index {1}.", value, i);
        }
        else if (~i == ar.Length) {
            Console.WriteLine("The object searched for, {0}, was ",
value);
            Console.WriteLine("not found,\nand no object in the array
had ");
            Console.WriteLine("greater value. ");
        }
        else {
            Console.WriteLine("The object searched for, {0}, was ",
value);
        }
    }
}
```

39

```
1         Console.Write("not found.\n\nThe next larger object is
2 at ");
3         Console.WriteLine("index {0}.", ~i);
4     }
5 }
6 }
7
```

8 The output is

9
10 The indices and elements of the array are:

11
12
13 [0]:0 [1]:2 [2]:4 [3]:6 [4]:8

14
15
16 The object searched for, 3, was not found.

17
18
19 The next larger object is at index 2.

20
21
22 The object searched for, 6, was found at index 3.

23
24
25 The object searched for, 9, was not found,

26
27
28 and no object in the array had greater value.
29

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

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

9 Summary

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

12 Parameters

13
14

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

15
16

16 Description

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

20 Exceptions

21
22

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

23
24
25

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

10 A **System.Object** that is a copy of the current instance.

11 Description

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

14 Behaviors

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

19 Default

20 As described above.

21 How and When to Override

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

23 Usage

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

25 Example

26

27 This example demonstrates the **System.Array.Clone** method.

28

29

```
[C#]
```

30

```
using System;
```

```

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

1
2
3
4
5

The elements of the cloned array are: 3 4 5

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

```
3 [ILASM]  
4 .method public hidebysig static void Copy(class  
5 System.Array sourceArray, class System.Array  
6 destinationArray, int32 length)  
  
7 [C#]  
8 public static void Copy(Array sourceArray, Array  
9 destinationArray, int length)
```

10 Summary

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

13 Parameters

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

16 Description

18 This version of **System.Array.Copy** is equivalent to
19 **System.Array.Copy** (*sourceArray*, *sourceArray.GetLowerBound(0)*,
20 *destinationArray*, *destinationArray.GetLowerBound(0)*, *length*).

21
22 If *sourceArray* and *destinationArray* are of different types,
23 **System.Array.Copy** performs widening conversions on the elements
24 of *sourceArray* as necessary before storing the information in
25 *destinationArray*. Value types will be boxed when being converted to a
26 **System.Object**. If the necessary conversion is a narrowing
27 conversion, a **System.ArrayTypeMismatchException** exception is
28 thrown. [Note: For information regarding valid conversions performed
29 by this method, see **System.Convert**.]

30
31 If an exception is thrown while copying, the state of *destinationArray*
32 is undefined.

33
34 If *sourceArray* and *destinationArray* are the same array,
35 **System.Array.Copy** copies the source elements safely to their
36 destination, as if the copy were done through an intermediate array.

1 **Exceptions**
 2
 3

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	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. -or- 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. -or- Both arrays are user-defined value types and are not of the same type.
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- $length < destinationArray.Length$.

4
 5 **Example**
 6

7 This example demonstrates the **System.Array.Copy** method.

8 [C#]
 9

```
10 using System;
11 public class ArrayCopyExample {
```

```
1         public static void Main() {
2             int[] intAryOrig = new int[3];
3             double[] dAryCopy = new double[3];
4             for (int i = 0; i < intAryOrig.Length; i++)
5                 intAryOrig[i] = i+3;
6             //copy the first 2 elements of the source into the
7 destination
8             Array.Copy(intAryOrig, dAryCopy, 2);
9             Console.WriteLine("The elements of the first array are:
10 ");
11             for (int i = 0; i < intAryOrig.Length; i++)
12                 Console.WriteLine("{0,3}", intAryOrig[i]);
13             Console.WriteLine();
14             Console.WriteLine("The elements of the copied array are:
15 ");
16             for (int i = 0; i < dAryCopy.Length; i++)
17                 Console.WriteLine("{0,3}", dAryCopy[i]);
18         }
19     }
20 }
```

21 The output is

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

24
25
26 The elements of the copied array are: 3 4 0
27

28

1 Array.Copy(System.Array, System.Int32, 2 System.Array, System.Int32, 3 System.Int32) Method

```
4 [ILASM]  
5 .method public hidebysig static void Copy(class  
6 System.Array sourceArray, int32 sourceIndex, class  
7 System.Array destinationArray, int32 destinationIndex,  
8 int32 length)
```

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

12 Summary

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

16 Parameters

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

19 Description

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

29
30 If an exception is thrown while copying, the state of *destinationArray*
31 is undefined.

32
33 If *sourceArray* and *destinationArray* are the same array,

1 **System.Array.Copy** copies the source elements safely to their
 2 destination as if the copy were done through an intermediate array.

3 **Exceptions**

4
 5

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 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> <p>$length < 0$.</p>
System.ArgumentException	$(sourceIndex + length) > (sourceArray.GetLowerBound(0) + sourceArray.Length)$.

```
(destinationIndex + length) >
(destinationArray.GetLowerBound(0) +
destinationArray.Length).
```

Example

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

[C#]

```
using System;
class ArrayCopyExample {
    public static void Main() {
        int[] intAry = { 0, 10, 20, 30, 40, 50 };
        Console.Write("The elements of the array are: ");
        foreach (int i in intAry)
            Console.Write("{0,3}", i);
        Console.WriteLine();
        Array.Copy(intAry, 2, intAry, 0, 4);
        Console.WriteLine("After copying elements 2 through 5
into elements 0 through 4");
        Console.Write("The elements of the array are: ");
        foreach (int i in intAry)
            Console.Write("{0,3}", i);
        Console.WriteLine();
    }
}
```

The output is

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

After copying elements 2 through 5 into elements 0 through
4

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

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

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

8 Summary

9 Copies all the elements of the current instance to the specified one-
10 dimensional array starting at the specified relative index in the
11 destination array.

12 Parameters

13
14

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

15
16

Description

17 [Note: This method is implemented to support the
18 **System.Collections.ICollection** interface. If implementing
19 **System.Collections.ICollection** is not explicitly required, use
20 **System.Array.Copy** to avoid an extra indirection.

21
22
23
24
25
26
27
28
29
30

index is a relative index, not the actual array index. If the index of *array* is zero-based, this value is the same as the actual index at which copying begins. If the lower bound of *array* is not zero, the value of *index* is added to the lower bound of *array* to get the actual index at which copying begins. For example, if the lower bound of *array* is 2 and the value of *index* is 1, the copying actually starts at index 3.

If this method throws an exception while copying, the state of *array* is undefined.]

31 Behaviors

32 As described above.

33 Default

34 As described above.

1 **How and When to Override**

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

4 **Usage**

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

7 **Exceptions**

8

9

Exception	Condition
System.ArgumentNullException	<i>array</i> is null .
System.RankException	The current instance has more than one dimension.
System.ArgumentOutOfRangeException	<i>index</i> < 0.
System.ArgumentException	<i>array</i> has more than one dimension. -or- <i>index</i> is greater than or equal to <i>array.Length</i> . -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 type of the current instance cannot be cast automatically to the type of <i>array</i> .

10

11 **Example**

12

13 The following example shows how to copy the elements of one
14 **System.Array** into another.

15

16

```
17           using System;  
18  
19           public class ArrayCopyToExample  
20           {  
21               public static void Main()  
22               {
```

```

1      Array aryOne = Array.CreateInstance(typeof(Object),
2      3);
3      aryOne.SetValue("one", 0);
4      aryOne.SetValue("two", 1);
5      aryOne.SetValue("three", 2);
6
7      Array aryTwo = Array.CreateInstance(typeof(Object),
8      5);
9      for (int i=0; i < aryTwo.Length; i++)
10         aryTwo.SetValue(i, i);
11
12         Console.WriteLine("The contents of the first array
13 are:");
14         foreach (object o in aryOne)
15             Console.Write("{0} ", o);
16         Console.WriteLine();
17         Console.WriteLine("The original contents of the
18 second array are:");
19         foreach (object o in aryTwo)
20             Console.Write("{0} ", o);
21         Console.WriteLine();
22
23         aryOne.CopyTo(aryTwo, 1);
24
25         Console.WriteLine("The new contents of the second
26 array are:");
27         foreach(object o in aryTwo)
28             Console.Write("{0} ", o);
29     }
30 }

```

31 The output is

32
33 The contents of the first array are:

34
35 one two three

36
37 The original contents of the second array are:

38
39 0 1 2 3 4

40
41 The new contents of the second array are:

1
2

0 one two three 4

3

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

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

9 Summary

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

12 Parameters

13
14

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

15
16
17

16 Return Value

18 A zero-based, one-dimensional **System.Array** object containing
19 *length* elements of type *elementType*.

20 Description

21 Reference-type elements will be set to **null**. Value-type elements will
22 be set to zero, except for **System.Boolean** elements, which will be
23 set to **false**.

24
25
26
27

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

28 Exceptions

29
30

Exception	Condition
System.ArgumentNullException	<i>elementType</i> is null .
System.ArgumentException	<i>elementType</i> is not a valid

	System.Type.
System.ArgumentOutOfRangeException	<i>length < 0.</i>

1
2
3

Example

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

6
7

[C#]

8
9

using System;

10

public class ArrayCreateInstanceExample

11

{

12

public static void Main()

13

{

14

15

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

16

for (int

17

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

18

intAry.SetValue(i*3,i);

19

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

20

foreach (int i in intAry)

21

Console.WriteLine("{0} ",i);

22

23

}

24

25

}

26

27

28

The output is

29

30

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

31

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

3 **Array.CreateInstance(System.Type,** 4 **System.Int32, System.Int32) Method**

```
5 [ILASM]  
6 .method public hidebysig static class System.Array  
7 CreateInstance(class System.Type elementType, int32  
8 length1, int32 length2)  
9 [C#]  
10 public static Array CreateInstance(Type elementType, int  
11 length1, int length2)
```

12 **Summary**

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

15 **Parameters**

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

18 **Return Value**

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

22 **Description**

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

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

1 **Exceptions**

2

3

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.

4

5 **Example**

6

7 The following example shows how to create and initialize a two-
8 dimensional **System.Array**.

9

10

[C#]

11

```
using System;
```

12

13

```
public class Create2DArrayExample
```

14

```
{
```

15

```
    public static void Main()
```

16

```
    {
```

17

```
        int i, j;
```

18

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

19

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

20

```
ary.GetUpperBound(0); i++)
```

21

```
    {
```

22

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

23

```
ary.GetUpperBound(1); j++)
```

24

```
    {
```

25

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

26

```
    }
```

27

```
    }
```

28

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

29

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

30

```
ary.GetUpperBound(0); i++)
```

31

```
    {
```

32

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

33

```
ary.GetUpperBound(1); j++)
```

34

```
    {
```

35

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

36

```
    }
```

37

```
    }
```

38

```
        Console.WriteLine();
```

39

```
    }
```

40

```
}
```

41

```
}
```

1

2

The output is

3

The elements of the array are:

4

0 1 2

5

10 11 12

6

20 21 22

7

30 31 32

8

40 41 42

9

10

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

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**.

1
2
3
4
5
6
7

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

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.

8
9
10

Example

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

11
12
13
14

[C#]

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

using System;

public class Create3DArrayExample
{
    public static void Main()
    {
        int i, j, k;
        Array ary = Array.CreateInstance(typeof(int), 2, 4,
3);
        for(i = ary.GetLowerBound(0); i <=
ary.GetUpperBound(0); i++)
        {
            for(j = ary.GetLowerBound(1); j <=
ary.GetUpperBound(1); j++)
            {
                for(k = ary.GetLowerBound(2); k <=
ary.GetUpperBound(2); k++)
                {
                    ary.SetValue((100*i + 10*j + k), i, j, k);
                }
            }
        }
    }
}

```

```

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

```

23 The output is

```

24 The elements of the array are:
25     0  1  2
26    10 11 12
27    20 21 22
28    30 31 32
29
30   100 101 102
31   110 111 112
32   120 121 122
33   130 131 132
34
35

```

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

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

```
5 [ILASM]  
6 .method public hidebysig static class System.Array  
7 CreateInstance(class System.Type elementType, class  
8 System.Int32[] lengths)  
9 [C#]  
10 public static Array CreateInstance(Type elementType, int[]  
11 lengths)
```

12 **Summary**

13 Creates a zero-based, multidimensional array of the specified
14 **System.Type** and dimension lengths.

15 **Parameters**

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

19 **Return Value**

21 A new zero-based, multidimensional **System.Array** instance of the
22 specified **System.Type** with the specified length for each dimension.
23 The **System.Array.Rank** of the new instance is equal to
24 *lengths.Length*.

25 **Description**

26 The number of elements in *lengths* is required to equal the number of
27 dimensions in the new **System.Array** instance. Each element of
28 *lengths* specifies the length of the corresponding dimension in the new
29 instance.

31 Reference-type elements will be set to **null**. Value-type elements will
32 be set to zero, except for **System.Boolean** elements, which will be
33 set to **false**.

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

4 **Exceptions**

5
 6

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.

7
 8
 9

8 **Example**

10 The following example shows how to create and initialize a
 11 multidimensional **System.Array**.

12
 13

[C#]

14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38

```

using System;

public class CreateMultiDimArrayExample
{
    public static void Main()
    {
        int i, j, k;
        int[] indexAry = {2, 4, 5};
        Array ary = Array.CreateInstance(typeof(int),
indexAry);
        for(i = ary.GetLowerBound(0); i <=
ary.GetUpperBound(0); i++)
        {
            for(j = ary.GetLowerBound(1); j <=
ary.GetUpperBound(1); j++)
            {
                for(k = ary.GetLowerBound(2); k <=
ary.GetUpperBound(2); k++)
                {
                    ary.SetValue((100*i + 10*j + k), i, j, k);
                }
            }
        }
        Console.WriteLine("The elements of the array are:");
    }
}

```

```

1         for(i = ary.GetLowerBound(0); i <=
2 ary.GetUpperBound(0); i++)
3     {
4         for(j = ary.GetLowerBound(1); j <=
5 ary.GetUpperBound(1); j++)
6     {
7         for(k = ary.GetLowerBound(2); k <=
8 ary.GetUpperBound(2); k++)
9     {
10            Console.WriteLine("{0, 3} ", ary.GetValue(i, j,
11 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 3 4
23 10 11 12 13 14
24 20 21 22 23 24
25 30 31 32 33 34
26
27 100 101 102 103 104
28 110 111 112 113 114
29 120 121 122 123 124
30 130 131 132 133 134
31

```

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

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

```
5 [ILASM]  
6 .method public hidebysig static class System.Array  
7 CreateInstance(class System.Type elementType, class  
8 System.Int32[] lengths, class System.Int32[] lowerBounds)  
9 [C#]  
10 public static Array CreateInstance(Type elementType, int[]  
11 lengths, int[] lowerBounds)
```

12 **Summary**

13 Creates a multidimensional array of the specified **System.Type** and
14 dimension lengths, with the specified lower bounds.

15 **Parameters**

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

18 **Return Value**

19 A new multidimensional **System.Array** of the specified **System.Type**
20 with the specified length and lower bound for each dimension.

21 **Description**

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

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

27 Each element of *lowerBounds* specifies the lower bound of the

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

10 **Exceptions**

11
 12

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</i> .Length = 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.

13
 14
 15

Example

16 The following example shows how to create and initialize a
 17 multidimensional **System.Array** with specified low bounds.

18
 19

[C#]

20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31

```
using System;

public class MultiDimNonZeroBoundExample
{
    public static void Main()
    {
        int i, j, k;
        int[] indexAry = {4, 2, 3};
        int[] lowboundAry = {3, 2, 1};
        Array ary = Array.CreateInstance(typeof(int),
        indexAry, lowboundAry);
```

```

1         for(i = ary.GetLowerBound(0); i <=
2 ary.GetUpperBound(0); i++)
3         {
4             for(j = ary.GetLowerBound(1); j <=
5 ary.GetUpperBound(1); j++)
6                 {
7                     for(k = ary.GetLowerBound(2); k <=
8 ary.GetUpperBound(2); k++)
9                         {
10                            ary.SetValue((100*i + 10*j + k), i, j, k);
11                        }
12                }
13        }
14        Console.WriteLine("The elements of the array are:");
15        for(i = ary.GetLowerBound(0); i <=
16 ary.GetUpperBound(0); i++)
17            {
18                for(j = ary.GetLowerBound(1); j <=
19 ary.GetUpperBound(1); j++)
20                    {
21                        for(k = ary.GetLowerBound(2); k <=
22 ary.GetUpperBound(2); k++)
23                            {
24                                Console.Write("{0, 3} ", ary.GetValue(i, j,
25 k));
26                            }
27                        Console.WriteLine();
28                    }
29                Console.WriteLine();
30            }
31    }
32 }
33

```

34 The output is

```

35 The elements of the array are:
36 321 322 323
37 331 332 333
38
39 421 422 423
40 431 432 433
41
42 521 522 523
43 531 532 533
44
45 621 622 623
46 631 632 633
47

```

1 `Array.GetEnumerator()` Method

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

7 **Summary**

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

9 **Return Value**

10

11 A **System.Collections.IEnumerator** for the current instance.

12 **Description**

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

15

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

20 **Behaviors**

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

28

29 The enumerator is in an invalid state if it is positioned before the first
30 element or after the last element of the current instance. Whenever
31 the enumerator is in an invalid state, a call to
32 **System.Collections.IEnumerator.Current** is required to throw a
33 **System.InvalidOperationException**.

34

35 **System.Collections.IEnumerator.Current** returns the same object
36 until either **System.Collections.IEnumerator.MoveNext** or
37 **System.Collections.IEnumerator.Reset** is called.

38

39 Once the enumerator of the current instance is moved immediately
40 past the last element of the current instance, subsequent calls to

1 **System.Collections.IEnumerator.MoveNext** return **false** and the
2 enumerator remains positioned immediately past the last element.

3 **Default**

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

5

6 [*Note:* For some multidimensional **System.Array** objects, it may be
7 desirable for an enumerator to process them in Column-major form.]

8 **How and When to Override**

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

10 **Usage**

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

12 **Example**

13

14 This example demonstrates the **System.Array.GetEnumerator**
15 method.

16

17

[C#]

18

```
using System;
```

19

```
using System.Collections;
```

20

```
public class ArrayGetEnumerator {
```

21

```
    public static void Main() {
```

22

```
        string[,] strAry = {{"1","one"}, {"2", "two"}, {"3",
```

23

```
"three"}}};
```

24

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

25

```
        IEnumerator sEnum = strAry.GetEnumerator();
```

26

```
        while (sEnum.MoveNext())
```

27

```
            Console.WriteLine(" {0}", sEnum.Current);
```

28

```
    }
```

29

```
}
```

30

31

The output is

32

33

The elements of the array are: 1 one 2 two 3 three

34

35

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

3 **Array.GetLowerBound(System.Int32)** 4 **Method**

```
5 [ILASM]  
6 .method public hidebysig instance int32 GetLowerBound(int32  
7 dimension)  
8 [C#]  
9 public int GetLowerBound(int dimension)
```

10 **Summary**

11 Returns the lower bound of the specified dimension in the current
12 instance.

13 **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.

17 **Return Value**

19 A **System.Int32** that contains the lower bound of the specified
20 dimension in the current instance.

21 **Description**

22 [Note: For example, **System.Array.GetLowerBound** (0) returns the
23 lower bound of the first dimension of the current instance, and
24 **System.Array.GetLowerBound(System.Array.Rank - 1)** returns
25 the lower bound of the last dimension of the current instance.]

26 **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

- 1
- 2
- 3

	current instance.
--	-------------------

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

3 **Array.GetUpperBound(System.Int32)** 4 **Method**

```
5 [ILASM]  
6 .method public hidebysig instance int32 GetUpperBound(int32  
7 dimension)  
8 [C#]  
9 public int GetUpperBound(int dimension)
```

10 **Summary**

11 Returns the upper bound of the specified dimension in the current
12 instance.

13 **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.

16 **Return Value**

17 A **System.Int32** that contains the upper bound of the specified
18 dimension in the current instance.

19 **Description**

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

24 **Exceptions**

Exception	Condition
System.IndexOutOfRangeException	<i>dimension</i> < 0. -or- <i>dimension</i> is equal to or greater than the

1
2
3

	System.Array.Rank property of the current instance.
--	--

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

3 **Array.GetValue(System.Int32[]) Method**

```
4 [ILASM]  
5 .method public hidebysig instance object GetValue(class  
6 System.Int32[] indices)  
  
7 [C#]  
8 public object GetValue(int[] indices)
```

9 **Summary**

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

12 **Parameters**

13
14

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

15
16
17

16 **Return Value**

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

20 **Description**

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

25
26
27
28

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

29 **Exceptions**

30
31

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

1
2
3

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.

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

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

7 Summary

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

10 Parameters

11
12

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

13
14
15

14 Return Value

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

18 Description

19 [Note: Use the **System.Array.GetLowerBound** and
20 **System.Array.GetUpperBound** methods to determine whether *index*
21 is out of bounds.]

22 Exceptions

23
24

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.

25
26
27

26 Example

1 This example demonstrates the **System.Array.GetValue** method.

2
3

[C#]

```
4 using System;
5 public class ArrayGetValueExample {
6     public static void Main() {
7         String[] strAry = { "one", "two", "three", "four",
8 "five" };
9         Console.WriteLine("The elements of the array are: ");
10        for(int i = 0; i < strAry.Length; i++)
11            Console.WriteLine(" '{0}' ", strAry.GetValue(i));
12    }
13 }
14
```

15 The output is

```
16
17 The elements of the array are: 'one' 'two' 'three' 'four'
18 'five'
```

19

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

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

```
5 [ILASM]  
6 .method public hidebysig instance object GetValue(int32  
7 index1, int32 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
12 instance.

13 Parameters

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

17 Return Value

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

21 Description

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

25 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 indexes for the

- 1
- 2
- 3

	corresponding dimension of the current instance.
--	--

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

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

```
5 [ILASM]  
6 .method public hidebysig instance object GetValue(int32  
7 index1, int32 index2, int32 index3)  
8 [C#]  
9 public object GetValue(int index1, int index2, int index3)
```

10 **Summary**

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

13 **Parameters**

14
15

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

16
17
18

17 **Return Value**

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

21 **Description**

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

25 **Exceptions**

26
27

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

1
2
3

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

```
3 [ILASM]  
4 .method public hidebysig static int32 IndexOf(class  
5 System.Array array, 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
10 index of the first occurrence of the specified **System.Object**.

11 Parameters

12
13

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

14
15
16

15 Return Value

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

21 Description

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

25
26 The elements will be compared using the semantics of the
27 **System.Object.Equals** method. For user-defined types,
28 **System.Object.Equals** is actually called.

29 Exceptions

30
31

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

1
2
3

Example

4 The following example demonstrates the **System.Array.IndexOf**
5 method.

6
7

[C#]

```
8       using System;
9       public class ArrayIndexOfExample {
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.IndexOf(intAry, 1);
17         Console.WriteLine("The first occurrence of 1 is at
18       index {0}", j);
19        }
20       }
```

21 The output is

22

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

24

25

26 The first occurrence of 1 is at index 1

27

28

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

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

9 Summary

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

13 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.

16 Return Value

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

24 Description

25 This version of **System.Array.IndexOf** is equivalent to
26 **System.Array.IndexOf** (*array*, *value*, *startIndex*, (*array*.Length -
27 *startIndex*)).

28
29 The elements will be compared using the semantics of the
30 **System.Object.Equals** method. For user-defined types,
31 **System.Object.Equals** is actually called.

32 Exceptions

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

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

- 1
- 2
- 3

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

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

11 Summary

12 Searches the specified one-dimensional **System.Array**, returning the
13 index of the first occurrence of the specified **System.Object** in the
14 specified range.

15 Parameters

16
17

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.
<i>count</i>	A System.Int32 that contains the number of elements to search, beginning with <i>startIndex</i> .

18
19
20

Return Value

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

26 Description

27 The elements will be compared using the semantics of the
28 **System.Object.Equals** method. For user-defined types,
29 **System.Object.Equals** is actually called.

1 Exceptions
2
3

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> < 0. -or- The sum of <i>count</i> and <i>startIndex</i> does not specify a valid range in <i>array</i> (i.e. <i>count</i> + <i>startIndex</i> > <i>array</i> .GetLowerBound(0) + <i>array</i> .Length).
System.RankException	<i>array</i> has more than one dimension.

4
5
6

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
10 by calling the 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
15 value type does not have a default constructor, the current instance is
16 not modified.

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

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

23

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

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

8 Summary

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

11 Parameters

12
13

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

14
15
16

15 Return Value

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

21 Description

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

25
26 The elements will be compared using the semantics of the
27 **System.Object.Equals** method. For user-defined types,
28 **System.Object.Equals** is actually called.

29 Exceptions

30
31

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

1
2
3

Example

4 The following example demonstrates the **System.Array.LastIndexOf**
5 method.

6
7

```
[C#]
```

8 using System;

9

```
10       public class ArrayLastIndexOfExample {  
11  
12           public static void Main() {  
13               int[] intAry = { 0, 1, 2, 0, 1 };  
14               Console.Write("The values of the array are: ");  
15               foreach(int i in intAry)  
16                   Console.Write("{0,5}", i);  
17               Console.WriteLine();  
18               int j = Array.LastIndexOf(intAry, 1);  
19               Console.WriteLine("The last occurrence of 1 is at  
20       index {0}", j);  
21               }  
22       }
```

23 The output is

24

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

26

27

28 The last occurrence of 1 is at index 4

29

30

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

```
[ILASM]
.method public hidebysig static int32 LastIndexOf(class
System.Array array, object value, int32 startIndex)

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

Summary

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

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.

Return Value

A **System.Int32** containing the index of the last occurrence of *value* in the range *startIndex* through the lower bound of *array*, 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

This version of **System.Array.LastIndexOf** is equivalent to **System.Array.LastIndexOf(array, value, startIndex, (array.Length - startIndex))**.

The elements will be compared using the semantics of the **System.Object.Equals** method. For user-defined types, **System.Object.Equals** is actually called.

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
- 3

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

```
4 [ILASM]  
5 .method public hidebysig static int32 LastIndexOf(class  
6 System.Array array, object value, int32 startIndex, int32  
7 count)
```

```
8 [C#]  
9 public static int LastIndexOf(Array array, object value,  
10 int startIndex, int count)
```

11 Summary

12 Searches the specified one-dimensional **System.Array**, returning the
13 index of the last occurrence of the specified **System.Object** in the
14 specified range.

15 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.
<i>count</i>	A System.Int32 that contains the number of elements to search, beginning with <i>startIndex</i> .

18

19 Return Value

20

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

26 Description

27 The elements will be compared using the semantics of the
28 **System.Object.Equals** method. For user-defined types,
29 **System.Object.Equals** is actually called.

1 Exceptions
2
3

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> < 0. -or- <i>count</i> and <i>startIndex</i> do not specify a valid range in <i>array</i> .
System.RankException	<i>array</i> has more than one dimension.

4
5
6

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

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

7 **Summary**

8 Reverses the sequence of the elements in the specified one-
9 dimensional **System.Array**.

10 **Parameters**

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

13 **Description**

15 This version of **System.Array.Reverse** is equivalent to
16 **System.Array.Reverse**(`array`, `array.GetLowerBound(0)`,
17 `array.Length`).

18 **Exceptions**

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

21
22
23

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

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

9 Summary

10 Reverses the sequence of the elements in the specified range of the
11 specified one-dimensional **System.Array**.

12 Parameters

13
14

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

15
16
17
18

Exceptions

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

19
20
21

Example

22 The following example demonstrates the **System.Array.Reverse**
23 method.

24
25

```
[C#]
```

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

16 The output is

17
18 The elements of the array are: one two three

19
20
21 After reversing the array,

22
23
24 the elements of the array are: three two one

25

26

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

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

8 Summary

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

11 Parameters

12
13

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.

14
15

Description

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

19
20
21

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

22 Exceptions

23
24

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

1
2
3

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

3 **Array.SetValue(System.Object,** 4 **System.Int32, System.Int32) Method**

```
5 [ILASM]  
6 .method public hidebysig instance void SetValue(object  
7 value, int32 index1, int32 index2)  
8 [C#]  
9 public void SetValue(object value, int index1, int index2)
```

10 **Summary**

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

13 **Parameters**

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

16 **Description**

17 [Note: For more information regarding valid conversions that will be
18 performed by this method, see **System.Convert**.
19
20

21 Use the **System.Array.GetLowerBound** and
22 **System.Array.GetUpperBound** methods to determine whether any
23 of the indices are out of bounds.]

24 **Exceptions**

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

1
2
3

	<i>value</i> is not assignment-compatible with the element type of the current instance.
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.

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

3 **Array.SetValue(System.Object,** 4 **System.Int32, System.Int32,** 5 **System.Int32) Method**

```
6 [ILASM]  
7 .method public hidebysig instance void SetValue(object  
8 value, int32 index1, int32 index2, int32 index3)
```

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

12 **Summary**

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

15 **Parameters**

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

19 **Description**

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

22
23 Use the **System.Array.GetLowerBound** and
24 **System.Array.GetUpperBound** methods to determine whether any
25 of the indices are out of bounds.]

26 **Exceptions**

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

1
2
3

System.ArgumentException	The current instance does not have exactly three dimensions. -or- <i>value</i> is not assignment-compatible with the element type of the current instance.
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.

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

3 **Array.SetValue(System.Object,** 4 **System.Int32[]) Method**

```
5 [ILASM]  
6 .method public hidebysig instance void SetValue(object  
7 value, class System.Int32[] indices)  
8 [C#]  
9 public void SetValue(object value, int[] indices)
```

10 **Summary**

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

13 **Parameters**

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

17 **Description**

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

22
23 [Note: For more information regarding valid conversions that will be
24 performed by this method, see **System.Convert**.

25
26 Use the **System.Array.GetLowerBound** and
27 **System.Array.GetUpperBound** methods to determine whether any
28 of the values in *indices* is out of bounds.]

29 **Exceptions**

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

1
2
3

System.ArgumentException	The number of dimensions in the current instance is not equal to the number of elements in <i>indices</i> . -or- <i>value</i> is not assignment-compatible with the element type of the current instance.
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.

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

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

7 Summary

8 Sorts the elements of the specified one-dimensional **System.Array**.

9 Parameters

10
11

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

12
13

Description

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

17
18
19
20

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

21 Exceptions

22
23

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

24
25
26

Example

27 This example demonstrates the **System.Array.Sort** method.

28
29

```
[C#]  
  
30 using System;  
31 public class ArraySortExample {
```

```
1         public static void Main() {
2             string[] strAry = { "All's", "well", "that", "ends",
3 "well" };
4             Console.Write("The original string array is: ");
5             foreach (String str in strAry)
6                 Console.Write(str + " ");
7             Console.WriteLine();
8             Array.Sort(strAry);
9             Console.Write("The sorted string array is: ");
10            foreach (string str in strAry)
11                Console.Write(str + " ");
12        }
13    }
```

14 The output is

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

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

20

21

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

2 Method

```
3 [ILASM]  
4 .method public hidebysig static void Sort(class  
5 System.Array keys, class 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
10 (one containing a set of keys and the other containing corresponding
11 items) based on the keys in the first specified **System.Array**.

12 Parameters

13
14

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

15
16

Description

17 This version of **System.Array.Sort** is equivalent to
18 **System.Array.Sort**(*keys*, *items*, *keys*.GetLowerBound(0),
19 *keys*.Length, **null**).

20

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

28

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

32 Exceptions

33
34

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.GetLowerBound(0)</i> does not equal <i>items.GetLowerBound(0)</i> . -or- <i>items</i> is not a null reference, and <i>keys.Length > items.Length</i> . -or- One or more elements in <i>keys</i> do not implement the System.IComparable interface.

1
2
3

Example

4
5
6

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

[C#]

```

7 using System;
8 public class ArraySortExample {
9     public static void Main() {
10         string[] strAry = { "All's", "well", "that", "ends",
11 "well" };
12         int[] intAry = { 3, 4, 0, 1, 2 };
13         Console.Write("The original string array is: ");
14         foreach (string str in strAry)
15             Console.Write(str + " ");
16         Console.WriteLine();
17         Console.Write("The key array is: ");
18         foreach (int i in intAry)
19             Console.Write(i + " ");
20         Console.WriteLine();
21         Array.Sort(intAry, strAry);
22         Console.Write("The sorted string array is: ");
23         foreach (string str in strAry)
24             Console.Write(str + " ");
25     }
26 }
```

```
1      The output is
2
3      The original string array is: All's well that ends well
4
5
6      The key array is: 3 4 0 1 2
7
8
9      The sorted string array is: that ends well All's well
10
```

11

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

```
3 [ILASM]  
4 .method public hidebysig static void Sort(class  
5 System.Array array, int32 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-
10 dimensional **System.Array**.

11 Parameters

12
13

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

14
15

Description

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

18
19
20
21
22

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

23 Exceptions

24
25

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> .

1
2
3

-or-

One or more elements in *array* do not implement the **System.IComparable** interface.

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

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

10 Summary

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

15 Parameters

Parameter	Description
<i>keys</i>	A one-dimensional System.Array that contains the keys to sort.
<i>items</i>	A one-dimensional System.Array 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 System.Int32 that contains the index at which sort begins.
<i>length</i>	A System.Int32 that contains the number of elements to sort.

18

19 Description

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

22

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

29

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

1 Exceptions
 2
 3

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	$index < keys.GetLowerBound(0)$. -or- $length < 0$.
System.ArgumentException	<i>items</i> is not a null reference, and $keys.GetLowerBound(0)$ does not equal $items.GetLowerBound(0)$. -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> . -or- One or more elements in <i>keys</i> do not implement the System.IComparable interface.

4
 5
 6

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

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

9 Summary

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

12 Parameters

13
14

Parameter	Description
<i>array</i>	The one-dimensional System.Array to sort.
<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

Description

17 This version of **System.Array.Sort** is equivalent to
18 **System.Array.Sort**(*array*, **null**, *array.GetLowerBound*(0),
19 *array.Length*, *comparer*).

20
21
22
23
24

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

25 Exceptions

26
27

Exception	Condition
System.ArgumentNullException	<i>array</i> is null .
System.RankException	<i>array</i> has more than one dimension.
System.ArgumentException	<i>comparer</i> is a null reference, and one or more elements in <i>array</i> do not implement the System.IComparable interface.

1
2
3

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

```
3 [ILASM]  
4 .method public hidebysig static void Sort(class  
5 System.Array keys, class System.Array items, class  
6 System.Collections.IComparer comparer)  
  
7 [C#]  
8 public static void Sort(Array keys, Array items, IComparer  
9 comparer)
```

10 Summary

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

15 Parameters

16
17

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

18

19 Description

20 This version of **System.Array.Sort** is equivalent to
21 **System.Array.Sort**(*keys*, *items*, *keys*.GetLowerBound(0),
22 *keys*.Length, *comparer*).

23

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

31

32 If *comparer* is a null reference, each element of *keys* is required to

1 implement the **System.IComparable** interface to be capable of
2 comparisons with every other element in *keys*.

3 **Exceptions**

4
5

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. -or- <i>comparer</i> is a null reference, and one or more elements in the <i>keys</i> do not implement the System.IComparable interface.

6
7
8

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

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

11 Summary

12 Sorts the elements in the specified section of the specified one-
13 dimensional **System.Array** using the specified
14 **System.Collections.IComparer** implementation.

15 Parameters

16
17

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

18

19 Description

20 This version of **System.Array.Sort** is equivalent to
21 **System.Array.Sort(array, null, index, length, comparer)**.

22

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

27 Exceptions

28

29

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

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

1
2
3

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  
6 System.Array keys, class System.Array items, int32 index,  
7 int32 length, class System.Collections.IComparer comparer)
```

```
8 [C#]  
9 public static void Sort(Array keys, Array items, int index,  
10 int length, IComparer comparer)
```

11 Summary

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

17 Parameters

18
19

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

20

21 Description

22 Each key in *keys* is required to have a corresponding item in *items*.
23 The sort is performed according to the order of *keys*. After a key is
24 repositioned during the sort, the corresponding item in *items* is
25 similarly repositioned. Only *keys.Length* elements of *items* will be
26 sorted. Therefore, *items* is sorted according to the arrangement of the
27 corresponding keys in *keys*. If the sort is not successfully completed,
28 the results are undefined.
29

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

4 **Exceptions**

5
 6

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> . -or- <i>comparer</i> is a null reference, and one or more elements in <i>keys</i> do not implement the System.IComparable interface.

7
 8
 9

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

2

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

8 **Summary**

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

11

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.
10 [Note: For more information, see **System.Collections.IList.Clear.**]

11

1 **Array.System.Collections.IList.Contains(S** 2 **ystem.Object) Method**

3 `[ILASM]`
4 `.method private final hidebysig virtual bool`
5 `System.Collections.IList.Contains(object value)`

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

8 **Summary**

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

12

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

2

3 [ILASM]
4 .method private final hidebysig virtual int32
5 System.Collections.IList.IndexOf(object value)

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

8 Summary

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

12

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

2

3 [ILASM]
4 .method private final hidebysig virtual void
5 System.Collections.IList.Insert(int32 index, object value)

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

8 Summary

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

11

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

2

3 [ILASM]
4 .method private final hidebysig virtual void
5 System.Collections.IList.Remove(object value)

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

8 Summary

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

12

1 **Array.System.Collections.IList.RemoveAt(2 System.Int32) Method**

3 [ILASM]
4 `.method private final hidebysig virtual void
5 System.Collections.IList.RemoveAt(int32 index)`

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

8 **Summary**

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

12

1 Array.IsFixedSize Property

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

7 Summary

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

11

1 Array.IsReadOnly Property

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

7 Summary

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

11

1 Array.IsSynchronized Property

```
2 [ILASM]
3 .property bool ICollection.IsSynchronized { public
4 hidebysig virtual abstract specialname bool
5 get_ICollection.IsSynchronized() }
6
7 [C#]
8 bool ICollection.IsSynchronized { get; }
```

8 Summary

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

12

1 Array.Length Property

```
2 [ILASM]  
3 .property int32 Length { public hidebysig specialname  
4 instance int32 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
9 instance.

10 Property Value

11

12 A **System.Int32** that contains the total number of elements in all the
13 dimensions of the current instance.

14 Description

15 This property is read-only.

16

1 Array.LongLength Property

2 [ILASM]

3 [C#]

4 `public long LongLength {get;}`

5 Summary

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

8 Property Value

9

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

11 Description

12 This property is read-only.

13

14 [*Note:* For additional information, see **System.Array.Length.**]

15

1 Array.Rank Property

```
2 [ILASM]  
3 .property int32 Rank { public hidebysig specialname  
4 instance int32 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

11 A **System.Int32** that contains the rank (number of dimensions) of the
12 current instance.

13 Description

14 This property is read-only.

15

1 Array.SyncRoot Property

```
2 [ILASM]
3 .property object ICollection.SyncRoot { public hideby sig
4 virtual abstract specialname object
5 get_ICollection.SyncRoot() }
6
7 [C#]
8 object ICollection.SyncRoot { get; }
```

8 Summary

9 Implemented to support the **System.Collections.ICollection**
10 interface. [Note: For more information, see
11 **System.Collections.ICollection.SyncRoot.**]

12

1 Array.System.Collections.ICollection.Count 2 Property

```
3 [ILASM]  
4 .property int32 ICollection.Count { public hidebySig  
5 virtual abstract specialname int32 get_ICollection.Count()  
6 }  
7 [C#]  
8 int ICollection.Count { get; }
```

9 Summary

10 Implemented to support the **System.Collections.ICollection**
11 interface. [Note: For more information, see
12 **System.Collections.ICollection.Count.**]

13

1 Array.System.Collections.IList.Item 2 Property

```
3 [ILASM]  
4 .property object IList.Item[int32 index] { public hidebysig  
5 virtual abstract specialname object get_IList.Item(int32  
6 index) public hidebysig virtual abstract specialname void  
7 set_IList.Item(int32 index, object value) }  
  
8 [C#]  
9 public virtual object this[int index] { get; set; }
```

10 Summary

11 Implemented to support the **System.Collections.IList** interface.
12 [Note: For more information, see **System.Collections.IList.Item**.]

13