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 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
  - CLSCompliantAttribute(true)

Implements:

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

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*.

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` and `System.Collections.Generic.IList<T>` interfaces. 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 non-negative `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 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.

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

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

Array() Constructor

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

[C#]
```
private Array()
```

Summary

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

The returned `IList<T>` has the same enumeration order as the array it wraps.

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Return Value	Description
The index of *value* in the array.	*value* was found.

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

[*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* - 1 if *array* is already sorted.]

Description

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.InvalidOperationException` 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.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.
System.ArgumentOutOfRangeException	*index* is less than *array*.`GetLowerBound(0)`. -or- *length* is less than zero.
System.ArgumentException	*index* + *length* is greater than *array*.`GetLowerBound(0)` + *array*.`Length`.

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

Example

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

[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.Write("[{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.Write( "The object searched for, {0}, was found ", value );
      Console.WriteLine( "at index {1}.", value, i );
    }
    else if ( ~i == ar.Length ) {
      Console.Write( "The object searched for, {0}, was ", value );
      Console.Write( "not found,\nand no object in the array had " );
      Console.WriteLine( "greater value. " );
    }
    else {
      Console.Write( "The object searched for, {0}, was ", value );
      Console.Write( "not found.\nThe next larger object is at " );
      Console.WriteLine( "index {0}.", ~i );
    }
  }
}
```

The output is

```
The indices and elements of the array are:
```

```
[0]:0 [1]:2 [2]:4 [3]:6 [4]:8


The object searched for, 3, was not found.


The next larger object is at index 2.


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


The object searched for, 9, was not found,


and no object in the array had greater value.
```

Array.BinarySearch(System.Array, System.Object, System.Collections.IComparer) Method

```
[ILAsm]
.method public hidebysig static int32 BinarySearch(class System.Array
array, object value, class System.Collections.IComparer comparer)

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

Summary

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

Parameters

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

Return Value

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

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

[*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.InvalidOperationException` 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.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.
System.InvalidOperationException	*comparer* is `null`, and both *value* and at least one element of *array* do not implement the `System.IComparable` interface.

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

```
[ILAsm]
.method public hidebysig static int32 BinarySearch(class System.Array
array, int32 index, int32 length, object value)

[C#]
public static int BinarySearch(Array array, int index, int length, object
value)
```

Summary

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

Parameters

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

Return Value

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

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

length).	array elements in the range of *index* to *index* + *length*- 1.

[*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* - 1 if *array* is already sorted.]

Description

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

value is compared to each element of *array* using the `System.IComparable` interface of the element being compared - or of *value* if the element being compared does not implement the interface - until an element with a value greater than or equal to *value* is found. 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.InvalidOperationException` 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.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.
System.ArgumentOutOfRangeException	*index* < *array*.GetLowerBound(0). -or- *length* < 0.
System.ArgumentException	*index* and *length* do not specify a valid range in *array* (i.e. *index* + *length* > *array*.GetLowerBound(0) + *array*.Length).

	-or- *array*.UpperBound == `System.Int32.MaxValue`.
System.InvalidOperationException	Either *value* or at least one element of *array* does not implement the `System.IComparable` interface.

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

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

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

Summary

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

Parameters

Parameter	Description
array	A `System.Array` to search for an object.
value	A `System.Object` for which to search.

Return Value

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

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

[*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.]

Description

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

value is compared to each element of *array* using the `System.IComparable` interface of the element being compared - or of *value* if the element being compared does not implement the interface - until an element with a value greater than or equal to *value* is found. 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.InvalidOperationException` 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.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.
System.InvalidOperationException	Both *value* and at least one element of *array* do not implement the `System.IComparable` interface.

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

```
[ILAsm]
.method public hidebysig static int32 BinarySearch<T>(!!0[] array, !!0
value)

[C#]
public static int BinarySearch<T>(T[] array, T value)
```

Summary

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

Parameters

Parameter	**Description**
array	The one-dimensional array to search.
value	The object for which to search.

Return Value

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

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

Description

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

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

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

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

A negative value, which is the bitwise complement of one more than the index of the final element.	*value* was not found, and *value* was greater than the value of all array elements.

Description

The comparer customizes how the elements are compared.

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

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

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

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static int32 BinarySearch<T>(!!0 array, int32
index, int32 length, !!0 value)

[C#]
public static int BinarySearch<T>(T[] array, int index, int length, T
value)
```

Summary

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

Parameters

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

Return Value

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

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

Description

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

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static int32 BinarySearch<T>(!!0[] array, int32
index, int32 length, !!0 value, class
System.Collections.Generic.IComparer`1<!!0> comparer)

[C#]
public static int BinarySearch<T>(T[] array, int index, int length, T
value, IComparer<T> comparer)
```

Summary

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

Parameters

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

Return Value

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

Return Value	Description

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

Description

The comparer customizes how the elements are compared.

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

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

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

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

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

Exceptions

Exception	**Condition**
System.ArgumentException	*index* and *length* do not specify a valid range in array.
System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*index* is less than zero -or-

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

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

```
[ILAsm]
.method public hidebysig static void Clear(class System.Array array, int32
index, int32 length)

[C#]
public static void Clear(Array array, int index, int length)
```

Summary

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

Parameters

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

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`.

Exceptions

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

Array.Clone() Method

[ILAsm]
```
.method public hidebysig virtual object Clone()
```

[C#]
```
public virtual object Clone()
```

Summary

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

Return Value

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

Description

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

Behaviors

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

Default

As described above.

How and When to Override

Override this method to return a clone of an array.

Usage

Use this method to obtain the clone of an array.

Example

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

[C#]

```
using System;
public class ArrayCloneExample {
  public static void Main() {
    int[] intAryOrig = { 3, 4, 5 };
    //must explicitly convert clones object into an array
    int[] intAryClone = (int[]) intAryOrig.Clone();
    Console.Write( "The elements of the first  array are: " );
    foreach( int i in intAryOrig )
      Console.Write( "{0,3}", i );
    Console.WriteLine();
    Console.Write( "The elements of the cloned array are: " );
    foreach( int i in intAryClone )
      Console.Write( "{0,3}", i );
    Console.WriteLine();
    //Clear the values of the original array.
    Array.Clear( intAryOrig, 0, 3 );
    Console.WriteLine( "After clearing the first array," );
    Console.Write( "The elements of the first  array are: " );
    foreach( int i in intAryOrig )
      Console.Write( "{0,3}", i );
    Console.WriteLine();
    Console.Write( "The elements of the cloned array are: " );
    foreach( int i in intAryClone )
      Console.Write( "{0,3}", i );
  }
}
```

The output is

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


The elements of the cloned array are: 3 4 5


After clearing the first array,


The elements of the first array are: 0 0 0


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

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

```
[ILAsm]
.method public hidebysig static !!1[] ConvertAll<T,U>(!!0[] array, class
System.Converter`1<!!0,!!1> converter)

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Description

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

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

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*sourceArray* or *destinationArray* is `null`.
System.RankException	*sourceArray* and *destinationArray* have different ranks.
System.ArrayTypeMismatchException	The elements in both arrays are built-in types, and converting from the type of the elements of *sourceArray* into the type of the elements in *destinationArray* 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 *sourceArray* is not assignment-compatible with the type of *destinationArray*.
System.ArgumentOutOfRangeException	*length* < 0.
System.ArgumentException	*length* > *sourceArray*.Length. -or- *length* > *destinationArray*.Length.

Example

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

[C#]

```
using System;
public class ArrayCopyExample {
   public static void Main() {
      int[] intAryOrig = new int[3];
      double[] dAryCopy = new double[3];
      for ( int i = 0; i < intAryOrig.Length; i++ )
         intAryOrig[i] = i+3;
      //copy the first 2 elements of the source into the destination
      Array.Copy( intAryOrig, dAryCopy, 2);
      Console.Write( "The elements of the first array are: " );
      for ( int i = 0; i < intAryOrig.Length; i++ )
         Console.Write( "{0,3}", intAryOrig[i] );
      Console.WriteLine();
      Console.Write( "The elements of the copied array are: " );
      for ( int i = 0; i < dAryCopy.Length; i++ )
         Console.Write( "{0,3}", dAryCopy[i] );
   }
}
```

The output is

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


The elements of the copied array are: 3 4 0
```

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

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

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

Summary

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

Parameters

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

Description

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

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*sourceArray* or *destinationArray* is `null`.
System.RankException	*sourceArray* and *destinationArray* have different ranks.
System.ArrayTypeMismatchException	The elements in both arrays are built-in types, and converting from the type of the elements of *sourceArray* into the type of the elements in *destinationArray* 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 element in *sourceArray* is assignment-incompatible with the type of *destinationArray*.
System.ArgumentOutOfRangeException	*sourceIndex* < *sourceArray*.GetLowerBound(0). -or- *destinationIndex* < *destinationArray*.GetLowerBound(0). -or-

	length < 0.
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
```

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

```
[ILAsm]
.method public hidebysig virtual void CopyTo(class System.Array array,
int32 index)

[C#]
public virtual void CopyTo(Array array, int index)
```

Summary

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

Parameters

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

Description

index is the array index in the destination array at which copying begins.

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

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

]

Behaviors

As described above.

Default

As described above.

How and When to Override

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

Usage

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

Exceptions

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

Example

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

[C#]

```
using System;

public class ArrayCopyToExample
{
   public static void Main()
   {
      Array aryOne = Array.CreateInstance(typeof(Object), 3);
      aryOne.SetValue("one", 0);
      aryOne.SetValue("two", 1);
      aryOne.SetValue("three", 2);

      Array aryTwo = Array.CreateInstance(typeof(Object), 5);
      for (int i=0; i < aryTwo.Length; i++)
         aryTwo.SetValue(i, i);

      Console.WriteLine("The contents of the first array are:");
      foreach (object o in aryOne)
         Console.Write("{0} ", o);
      Console.WriteLine();
      Console.WriteLine("The original contents of the second array are:");
      foreach (object o in aryTwo)
         Console.Write("{0} ", o);
      Console.WriteLine();

      aryOne.CopyTo(aryTwo, 1);

      Console.WriteLine("The new contents of the second array are:");
      foreach( object o in aryTwo)
         Console.Write("{0} ", o);
   }
}
```

The output is

```
The contents of the first array are:

one two three

The original contents of the second array are:

0 1 2 3 4

The new contents of the second array are:

0 one two three 4
```

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

Exceptions

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

Example

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

[C#]

```
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:");
      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++ )
            {
               Console.Write("{0, 3} ", ary.GetValue(i, j, k));
            }
            Console.WriteLine();
            }
            Console.WriteLine();
        }
    }
}
```

The output is

```
The elements of the array are:
  0   1   2   3   4
 10  11  12  13  14
 20  21  22  23  24
 30  31  32  33  34

100 101 102 103 104
110 111 112 113 114
120 121 122 123 124
130 131 132 133 134
```

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
elementType	The `System.Type` of the elements contained in the new `System.Array` instance.
length1	A `System.Int32` that contains the number of elements contained in the first dimension of the new `System.Array` instance.
length2	A `System.Int32` that contains the number of elements contained in the second dimension of the new `System.Array` instance.
length3	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`.

[*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	*elementType* is `null`.
System.ArgumentException	*elementType* is not a valid `System.Type`.
System.ArgumentOutOfRangeException	*length1* < 0. -or- *length2* < 0. -or- *length3* < 0.

Example

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

[C#]

```
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 );
```

```
              }
           }
        }
        Console.WriteLine("The elements of the array are:");
        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++ )
              {
                 Console.Write("{0, 3} ", ary.GetValue(i, j, k));
              }
              Console.WriteLine();
           }
           Console.WriteLine();
        }
     }
  }
```

The output is

```
The elements of the array are:
  0   1   2
 10  11  12
 20  21  22
 30  31  32

100 101 102
110 111 112
120 121 122
130 131 132
```

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*elementType* is `null`.
System.ArgumentException	*elementType* is not a valid `System.Type`.
System.ArgumentOutOfRangeException	*length1* < 0. -or- *length2* < 0.

Example

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

[C#]

```
using System;

public class Create2DArrayExample
{
   public static void Main()
   {
      int i, j;
      Array ary = Array.CreateInstance( typeof(int), 5, 3 );
      for( i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++ )
      {
         for( j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++ )
         {
            ary.SetValue( (10*i + j), i, j );
         }
      }
      Console.WriteLine("The elements of the array are:");
      for( i = ary.GetLowerBound(0); i <= ary.GetUpperBound(0); i++)
      {
         for( j = ary.GetLowerBound(1); j <= ary.GetUpperBound(1); j++)
         {
            Console.Write("{0, 2} ", ary.GetValue(i, j));
         }
         Console.WriteLine();
      }
   }
}
```

The output is

```
The elements of the array are:
 0  1  2
10 11 12
20 21 22
30 31 32
40 41 42


```

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

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

Exceptions

Exception	Condition

System.ArgumentNullException	*elementType* is `null`.
System.ArgumentException	*elementType* is not a valid `System.Type`.
System.ArgumentOutOfRangeException	*length* < 0.

Example

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

[C#]

```
using System;

public class ArrayCreateInstanceExample
{

   public static void Main()
   {

      Array intAry = Array.CreateInstance(typeof(int),5);
      for (int i=intAry.GetLowerBound(0);i<=intAry.GetUpperBound(0);i++)
         intAry.SetValue(i*3,i);
      Console.Write("The values of the array are:");
      foreach (int i in intAry)
         Console.Write("{0} ",i);

   }

}
```

The output is

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

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

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

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

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

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

Exceptions

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

Example

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

[C#]

```
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 );
      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:");
         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++ )
               {
                  Console.Write("{0, 3} ", ary.GetValue(i, j, k));
               }
               Console.WriteLine();
            }
            Console.WriteLine();
         }
      }
   }
```

The output is

```
The elements of the array are:
321 322 323
331 332 333

421 422 423
431 432 433

521 522 523
531 532 533

621 622 623
631 632 633
```

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

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

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

Summary

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

Parameters

Parameter	**Description**
array	The array to search.
match	The predicate that defines the conditions of the elements to search for.

Return Value

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

Description

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

[C#]
public static T[] FindAll<T>(T[] array, Predicate<T> match)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* or *match* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than zero or greater than

	array`.Length`.

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition

System.ArgumentNullException	*array* or *match* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than zero. -or- *count* is less than zero. -or- *startIndex* + *count* is greater than *array*.`Length`.

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* or *match* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than zero or greater than

	array`.Length`.

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition

System.ArgumentNullException	*array* or *match* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than zero or greater than *array*.`Length`. -or- *count* is less than zero. -or- *count* is greater than *startIndex* + 1.

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

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

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

Summary

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

Parameters

Parameter	**Description**
array	The array on whose elements the action is to be performed.
action	The action to perform on each element of *array*.

Description

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

Exceptions

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

Array.GetEnumerator() Method

```
[ILAsm]
.method public hidebysig virtual class System.Collections.IEnumerator
GetEnumerator()

[C#]
public virtual IEnumerator GetEnumerator()
```

Summary

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

Return Value

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

Description

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

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

Behaviors

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

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

`System.Collections.IEnumerator.Current` returns the same object until either `System.Collections.IEnumerator.MoveNext` or `System.Collections.IEnumerator.Reset` is called. `System.Collections.IEnumerator.MoveNext` sets `System.Collections.IEnumerator.Current` to the next element.

If `System.Collections.IEnumerator.MoveNext` passes the end of the collection, the enumerator is positioned after the last element in the collection and `System.Collections.IEnumerator.MoveNext` returns false. When the enumerator is at this position, subsequent calls to`System.Collections.IEnumerator.MoveNext` also return `false`. If the last call to `System.Collections.IEnumerator.MoveNext` returned `false`, `System.Collections.IEnumerator.Current` is unspecified. To set `System.Collections.IEnumerator.Current` to the first element of the collection again, you can call `System.Collections.IEnumerator.Reset` followed by `System.Collections.IEnumerator.MoveNext`.

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

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

Default

Multidimensional arrays will be processed in Row-major form.

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

How and When to Override

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

Usage

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

Example

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

[C#]

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

```
    }
}
```

The output is

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

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

Array.GetLength(System.Int32) Method

```
[ILAsm]
.method public hidebysig int32 GetLength(int32 dimension)

[C#]
public int GetLength(int dimension)
```

Summary

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

Parameters

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

Return Value

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

Exceptions

Exception	Condition
System.IndexOutOfRangeException	*dimension* is less than zero. -or- *dimension* is equal to or greater than `System.Array.Rank`.

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

Array.GetLowerBound(System.Int32) Method

```
[ILAsm]
.method public hidebysig instance int32 GetLowerBound(int32 dimension)

[C#]
public int GetLowerBound(int dimension)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

	instance.

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

Array.GetUpperBound(System.Int32) Method

```
[ILAsm]
.method public hidebysig instance int32 GetUpperBound(int32 dimension)

[C#]
public int GetUpperBound(int dimension)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

	instance.

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

Array.GetValue(System.Int32[]) Method

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

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

Summary

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

Parameters

Parameter	Description
indices	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.

Return Value

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

Description

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*indices* is `null`.
System.ArgumentException	The number of dimensions in the current instance is not equal to the number of elements in *indices*.

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

Array.GetValue(System.Int32) Method

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

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

Summary

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

Parameters

Parameter	Description
index	A `System.Int32` that contains the position of the value to get from the current instance.

Return Value

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

Description

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

Exceptions

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

Example

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

[C#]

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

The output is

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

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition

System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than *array*.`GetLowerBound(0)`. -or- *count* is less than zero. -or- *startIndex* + *count* is greater than *array*.`GetLowerBound(0)` + *array*.`Length`.
System.RankException	*array* has more than one dimension.

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

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

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

Summary

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

Parameters

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

Return Value

A `System.Int32` containing the index of the first occurrence of *value* in *array*, within the range *startIndex* through the last element 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.IndexOf` is equivalent to `System.Array.IndexOf` (*array*, *value*, *startIndex*, (*array*.Length - *startIndex*+*array*.GetLowerBound(0))).

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

Exceptions

Exception	**Condition**

System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than *array*`.GetLowerBound(0)` or greater than *array*`.GetLowerBound(0)` + *array*`.Length`.
System.RankException	*array* has more than one dimension.

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

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

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

Summary

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

Parameters

Parameter	Description
array	A one-dimensional `System.Array` to search.
value	A `System.Object` to locate in *array*.

Return Value

A `System.Int32` containing the index of the first occurrence of *value* in *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 a failed search.]

Description

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.

Example

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

[C#]

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

The output is

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


The first occurrence of 1 is at index 1
```

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

```
[ILAsm]
.method public hidebysig static int32 IndexOf<T>(!!0[] array, !!0 value,
int32 startIndex, int32 count)

[C#]
public static int IndexOf<T>(T[] array, T value, int startIndex, int
count)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition

System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than zero. -or- *count* is less than zero. -or- *startIndex* + *count* is greater than `System.Array.Length`.

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

```
[ILAsm]
.method public hidebysig static int32 IndexOf<T>(!!0[] array, !!0 value,
int32 startIndex)

[C#]
public static int IndexOf<T>(T[] array, T value, int startIndex)
```

Summary

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

Parameters

Parameter	**Description**
array	The array to search.
value	The value to locate.
startIndex	The zero-based starting index of the search.

Return Value

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

Description

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

Exceptions

Exception	**Condition**
System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is less than zero or greater than

	array`.Length`.

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

```
[ILAsm]
.method public hidebysig static int32 IndexOf<T>(!!0[] array, !!0 value)

[C#]
public static int IndexOf<T>(T[] array, T value)
```

Summary

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

Parameters

Parameter	**Description**
array	The array to search.
value	The value to locate.

Return Value

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

Description

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

Exceptions

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

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

Array.Initialize() Method

[ILAsm]
```
.method public hidebysig instance void Initialize()
```

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

Summary

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

Description

This method cannot be used on reference-type arrays.

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

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

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

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

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

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

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

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

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
array	A one-dimensional `System.Array` to search.
value	A `System.Object` to locate in *array*.
startIndex	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*,*startIndex*+ 1 - *array*.GetLowerBound(0)).

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is outside the range of valid indices for *array*.
System.RankException	*array* has more than one dimension.

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

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

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

Summary

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

Parameters

Parameter	Description
array	A one-dimensional `System.Array` to search.
value	A `System.Object` to locate in *array*.

Return Value

A `System.Int32` containing the index of the last occurrence in *array* of *value*, 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*, (*array*.GetLowerBound(0) + *array*.Length - 1), *array*.Length).

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

Exceptions

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

System.RankException	*array* has more than one dimension.

Example

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

[C#]

```
using System;

public class ArrayLastIndexOfExample {

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

The output is

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


The last occurrence of 1 is at index 4
```

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

```
[ILAsm]
.method public hidebysig static int32 LastIndexOf<T>(!!0[] array, !!0
value, int32 startIndex, int32 count)

[C#]
public static int LastIndexOf<T>(T[] array, T value, int startIndex, int
count)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition

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

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

```
[ILAsm]
.method public hidebysig static int32 LastIndexOf<T>(!!0[] array, !!0
value, int32 startIndex)

[C#]
public static int LastIndexOf<T>(T[] array, T value, int startIndex)
```

Summary

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

Parameters

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

Return Value

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

Description

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`.
System.ArgumentOutOfRangeException	*startIndex* is outside the range of valid indices for *array*.

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

[ILAsm]
```
.method public hidebysig static int32 LastIndexOf<T>(!!0[] array, !!0
value)
```

[C#]
```
public static int LastIndexOf<T>(T[] array, T value)
```

Summary

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

Parameters

Parameter	Description
array	The array to search.
value	The value to locate.

Return Value

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

Description

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static void Resize<T>(!!0[]& array, int32
newSize)

[C#]
public static void Resize<T>(ref T[] array, int newSize)
```

Summary

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

Parameters

Parameter	Description
array	The array to resize. -or- `null` to create a new array with the specified size.
newSize	The size of the new array.

Description

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static void Reverse(class System.Array array, int32 index, int32 length)

[C#]
public static void Reverse(Array array, int index, int length)
```

Summary

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

Parameters

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

Exceptions

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

Example

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

[C#]

```
using System;
public class ArrayReverseExample {
   public static void Main() {
      string[] strAry = { "one", "two", "three" };
      Console.Write( "The elements of the array are:");
      foreach( string str in strAry )
         Console.Write( " {0}", str );
      Array.Reverse( strAry );
      Console.WriteLine();
      Console.WriteLine( "After reversing the array," );
      Console.Write( "the elements of the array are:");
      foreach( string str in strAry )
         Console.Write( " {0}", str );
   }
}
```

The output is

```
The elements of the array are: one two three


After reversing the array,


the elements of the array are: three two one
```

Array.Reverse(System.Array) Method

```
[ILAsm]
.method public hidebysig static void Reverse(class System.Array array)

[C#]
public static void Reverse(Array array)
```

Summary

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

Parameters

Parameter	Description
array	The one-dimensional `System.Array` to reverse.

Description

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.

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

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

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

Summary

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

Parameters

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

Description

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

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

]

Exceptions

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

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

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

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

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

Summary

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

Parameters

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

Description

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

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

]

Exceptions

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

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

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

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

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

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

Summary

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

Parameters

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

Description

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

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

]

Exceptions

Exception	Condition

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

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

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

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

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

Summary

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

Parameters

Parameter	Description
value	A `System.Object` that contains the new value for the specified element.
indices	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.

Description

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

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

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

]

Exceptions

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

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

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

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

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

Summary

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

Parameters

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

Description

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static void Sort(class System.Array array, int32
index, int32 length, class System.Collections.IComparer comparer)

[C#]
public static void Sort(Array array, int index, int length, IComparer
comparer)
```

Summary

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

Parameters

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

Description

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

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.

Exceptions

Exception	Condition

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

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

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

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

Summary

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

Parameters

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

Description

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

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

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

Exceptions

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

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

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

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

Summary

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

Parameters

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

Description

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

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.

Exceptions

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

	not implement the `System.IComparable` interface.

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

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

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

Summary

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

Parameters

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

Description

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

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

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

Exceptions

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

Array.Sort(System.Array) Method

```
[ILAsm]
.method public hidebysig static void Sort(class System.Array array)

[C#]
public static void Sort(Array array)
```

Summary

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

Parameters

Parameter	Description
array	A one-dimensional `System.Array` to sort.

Description

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

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

Exceptions

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

Example

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

[C#]

```
using System;
public class ArraySortExample {
   public static void Main() {
      string[] strAry = { "All's", "well", "that", "ends", "well" };
      Console.Write( "The original string array is: " );
      foreach ( String str in strAry )
         Console.Write( str + " " );
      Console.WriteLine();
      Array.Sort( strAry );
      Console.Write( "The sorted string array is: " );
      foreach ( string str in strAry )
         Console.Write( str + " " );
   }
}
```

The output is

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


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

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

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

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

Summary

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

Parameters

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

Description

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

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

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

Exceptions

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

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

Example

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

[C#]

```
using System;
public class ArraySortExample {
   public static void Main() {
      string[] strAry = { "All's", "well", "that", "ends", "well" };
      int[] intAry = { 3, 4, 0, 1, 2 };
      Console.Write( "The original string array is: " );
      foreach ( string str in strAry )
         Console.Write( str + " " );
      Console.WriteLine();
      Console.Write( "The key array is: " );
      foreach ( int i in intAry )
         Console.Write( i + " " );
      Console.WriteLine();
      Array.Sort( intAry, strAry );
      Console.Write( "The sorted string array is: " );
      foreach ( string str in strAry )
         Console.Write( str + " " );
   }
}
```

The output is

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

The key array is: 3 4 0 1 2


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


```

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

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

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

Summary

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

Parameters

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

Description

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

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.

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`.
System.RankException	*array* has more than one dimension.
System.ArgumentOutOfRangeException	*index* < *array*.GetLowerBound(0). -or-

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

Array.Sort<K,V>(K[], V[], System.Int32, System.Int32, System.Collections.Generic.IComparer<K>) Method

```
[ILAsm]
.method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items, int32 index, int32 length, class System.Collections.Generic.IComparer`1<!!0> comparer)

[C#]
public static void Sort<K,V>(K[] keys, V[] items, int index, int length, IComparer<K> comparer)
```

Summary

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

Parameters

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

Description

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

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

Exceptions

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

Array.Sort<K,V>(K[], V[], System.Collections.Generic.IComparer<K>) Method

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

[C#]
public static void Sort<K,V>(K[] keys, V[] items, IComparer<K> comparer)
```

Summary

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

Parameters

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

Description

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

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items,
int32 index, int32 length)

[C#]
public static void Sort<K,V>(K[] keys, V[] items, int index, int length)
```

Summary

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

Parameters

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

Description

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

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

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

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

Exceptions

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

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

[ILAsm]
```
.method public hidebysig static void Sort<K,V>(!!0[] keys, !!1[] items)
```

[C#]
```
public static void Sort<K,V>(K[] keys, V[] items)
```

Summary

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

Parameters

Parameter	Description
keys	The array that contains the keys to sort.
items	The array that contains the items that correspond to each of the keys in *keys*. -or- `null` to sort only the *keys* array.

Description

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

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

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

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

Exceptions

Exception	Condition
System.ArgumentException	*items* is not `null`, and the length of *keys* does not

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

Array.Sort<T>(T[], System.Int32, System.Int32, System.Collections.Generic.IComparer<T>) Method

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

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

Summary

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

Parameters

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

Description

If *comparer* is null, each element within the specified range of elements in *array* must 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 undefined.

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

Exceptions

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

Array.Sort<T>(T[], System.Collections.Generic.IComparer<T>) Method

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

[C#]
public static void Sort<T>(T[] array, IComparer<T> comparer)
```

Summary

Sorts the elements in an array using the specified comparer.

Parameters

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

Description

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

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

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

Exceptions

Exception	Condition

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

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

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

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

Summary

Sorts the elements in an array using the specified comparison.

Parameters

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

Description

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

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

Exceptions

Exception	Condition
System.ArgumentNullException	*array* is `null`. -or- *comparison* is `null`.

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

```
[ILAsm]
.method public hidebysig static void Sort<T>(!!0[] array)

[C#]
public static void Sort<T>(T[] array)
```

Summary

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

Parameters

Parameter	Description
array	The array to sort.

Description

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

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

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

Exceptions

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

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

```
[ILAsm]
.method public hidebysig static void Sort<T>(!!0[] array, int32 index,
int32 length)

[C#]
public static void Sort<T>(T[] array, int index, int length)
```

Summary

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

Parameters

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

Description

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

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

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

Exceptions

Exception	Condition
System.ArgumentException	*index* and *length* do not specify a valid range in *array*.
System.ArgumentNullException	*array* is `null`.

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

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

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

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

Summary

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

Array.System.Collections.IList.Clear() Method

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

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

Summary

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

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

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

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

Summary

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

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

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

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

Summary

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

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

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

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

Summary

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

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

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

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

Summary

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

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

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

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

Summary

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

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

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

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

Summary

Determines whether every element in the array matches the predicate.

Parameters

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

Return Value

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

Description

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

Exceptions

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

Array.IsFixedSize Property

[ILAsm]
```
.property public bool IsFixedSize { public hidebysig virtual abstract
specialname bool get_IsFixedSize() }
```

[C#]
```
public bool IsFixedSize { get; }
```

Summary

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

Array.IsReadOnly Property

[ILAsm]
```
.property public bool IsReadOnly { public hidebysig virtual abstract
specialname bool get_IsReadOnly() }
```

[C#]
```
public bool IsReadOnly { get; }
```

Summary

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

Array.IsSynchronized Property

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

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

Summary

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

Array.Length Property

[ILAsm]
```
.property int32 Length { public hidebysig specialname instance int32
get_Length() }
```

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

Summary

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

Property Value

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

Description

This property is read-only.

Array.LongLength Property

[ILAsm]
```
.property int64 Length { public hidebysig specialname instance int64
get_LongLength() }
```

[C#]
```
public long LongLength {get;}
```

Summary

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

Property Value

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

Description

This property is read-only.

Array.Rank Property

[ILAsm]
```
.property int32 Rank { public hidebysig specialname instance int32
get_Rank() }
```

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

Summary

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

Property Value

A `System.Int32` that contains the rank (number of dimensions) of the current instance.

Description

This property is read-only.

Array.SyncRoot Property

[ILAsm]
```
.property public object SyncRoot { public hidebysig virtual abstract
specialname object get_SyncRoot() }
```

[C#]
```
public object SyncRoot { get; }
```

Summary

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

Array.System.Collections.ICollection.Count Property

```
[ILAsm]
.property int32 ICollection.Count { public hidebysig virtual abstract
specialname int32 get_ICollection.Count() }

[C#]
int ICollection.Count { get; }
```

Summary

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

Array.System.Collections.IList.Item Property

[ILAsm]
```
.property object IList.Item[int32 index] { public hidebysig virtual
abstract specialname object get_IList.Item(int32 index) public hidebysig
virtual abstract specialname void set_IList.Item(int32 index, object
value) }
```

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

Summary

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