

1 System.Runtime.InteropServices.Marshal

2 Class

```
3 [ILAsm]  
4 .class public abstract sealed Marshal extends System.Object  
5 [C#]  
6 public static class Marshal
```

7 Assembly Info:

- 8 • *Name:* mscorlib
- 9 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- 10 • *Version:* 4.0.0.0
- 11 • *Attributes:*
 - 12 ○ CLSCompliantAttribute(true)

13 Summary

14 Provides a collection of methods for allocating unmanaged memory, copying unmanaged
15 memory blocks, and converting managed to unmanaged types, as well as other
16 miscellaneous methods used when interacting with unmanaged code.

17 Inherits From: System.Object

18

19 **Library:** RuntimeInfrastructure

20

21 Description

22 The static methods defined on the System.Runtime.InteropServices.Marshal class
23 are essential to working with unmanaged code. Most methods defined in this class are
24 typically used by developers who want to provide a bridge between the managed and
25 unmanaged programming models. For example, the
26 System.Runtime.InteropServices.Marshal.StringToHGlobalAnsi method copies
27 ANSI characters from a specified string (in the managed heap) to a buffer in the
28 unmanaged heap. It also allocates the target heap of the right size.

29

30 The Read and Write methods in the System.Runtime.InteropServices.Marshal class
31 support both aligned and unaligned access.

32

1 Marshal.SystemDefaultCharSize Field

```
2 [ILAsm]  
3 .field public static initonly int32 SystemDefaultCharSize  
4 [C#]  
5 public static readonly int SystemDefaultCharSize
```

6 Summary

7 Represents the default character size on the system; the default is 2 for Unicode
8 systems and 1 for ANSI or UTF-8 based systems. This field is read-only.

9

1 Marshal.SystemMaxDBCSCharSize Field

```
2 [ILAsm]  
3 .field public static initonly int32 SystemMaxDBCSCharSize  
4 [C#]  
5 public static readonly int SystemMaxDBCSCharSize
```

6 Summary

7 Represents the maximum size of a double byte character set (DBCS) size, in bytes, for
8 the current operating system. This field is read-only.

9

1 Marshal.AllocHGlobal(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig static native int AllocHGlobal(int32 cb) cil  
4 managed  
5 [C#]  
6 public static IntPtr AllocHGlobal (int cb)
```

7 Summary

8 Allocates the specified number of bytes from the unmanaged memory of the process.

9 Parameters

Parameter	Description
<i>cb</i>	The required number of bytes in memory.

10

11 Return Value

12 A pointer to the newly allocated memory. This memory must be released using the
13 `System.Runtime.InteropServices.Marshal.FreeHGlobal` method.

14 Description

15 `System.Runtime.InteropServices.Marshal.AllocHGlobal` is the memory allocation
16 method in the `System.Runtime.InteropServices.Marshal` class.

17

18 The allocated memory is not zero-filled.

19 Exceptions

Exception	Condition
System.OutOfMemoryException	There is insufficient memory to satisfy the request.

20

21

1 **Marshal.AllocHGlobal(System.IntPtr) Method**

```
2 [ILAsm]
3 .method public hidebysig static native int AllocHGlobal(native int cb) cil
4 managed
5 [C#]
6 public static IntPtr AllocHGlobal (IntPtr cb)
```

7 **Summary**

8 Allocates the specified number of bytes from the unmanaged memory of the process.

9 **Parameters**

Parameter	Description
<i>cb</i>	The required number of bytes in memory.

10 **Return Value**

11 A pointer to the newly allocated memory. This memory must be released using the
12 `System.Runtime.InteropServices.Marshal.FreeHGlobal` method.

13 **Description**

14 `System.Runtime.InteropServices.Marshal.AllocHGlobal` is the allocation method in
15 the `System.Runtime.InteropServices.Marshal` class.

16 The allocated memory is not zero-filled.

17 **Exceptions**

Exception	Condition
System.OutOfMemoryException	There is insufficient memory to satisfy the request.

18
19
20
21

1 Marshal.Copy(System.Byte[], System.Int32, 2 System.IntPtr, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(uint8[] source, int32  
5 startIndex, native int destination, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (byte[] source, int startIndex, IntPtr  
8 destination, int length)
```

9 Summary

10 Copies data from a one-dimensional, managed 8-bit unsigned integer array to an
11 unmanaged memory pointer.

12 Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 You can use this method to copy a subset of a one-dimensional managed array to an
16 unmanaged C-style array.

17 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>source</i> , <i>startIndex</i> , <i>destination</i> , or <i>length</i> is null.

18
19

1 Marshal.Copy(System.Char[], System.Int32, 2 System.IntPtr, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(char[] source, int32 startIndex,  
5 native int destination, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (char[] source, int startIndex, IntPtr  
8 destination, int length)
```

9 Summary

10 Copies data from a one-dimensional, managed character array to an unmanaged
11 memory pointer.

12 Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 You can use this method to copy a subset of a one-dimensional managed array to an
16 unmanaged C-style array.

17 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>startIndex</i> , <i>destination</i> , or <i>length</i> is null.

18
19

1 Marshal.Copy(System.Double[], 2 System.Int32, System.IntPtr, System.Int32) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig static void Copy(float64[] source, int32  
6 startIndex, native int destination, int32 length) cil managed  
  
7 [C#]  
8 public static void Copy (double[] source, int startIndex, IntPtr  
9 destination, int length)
```

10 Summary

11 Copies data from a one-dimensional, managed double-precision floating-point number
12 array to an unmanaged memory pointer.

13 Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

14 15 Description

16 You can use this method to copy a subset of a one-dimensional managed array to an
17 unmanaged C-style array.

18 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>source</i> , <i>startIndex</i> , <i>destination</i> , or <i>length</i> is null.

19

Marshal.Copy(System.Int16[], System.IntPtr, System.Int32, System.IntPtr, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Copy(int16[] source, int32
startIndex, native int destination, int32 length) cil managed

[C#]
public static void Copy (short[] source, int startIndex, IntPtr
destination, int length)
```

Summary

Copies data from a one-dimensional, managed 16-bit signed integer array to an unmanaged memory pointer.

Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

Description

You can use this method to copy a subset of a one-dimensional managed array to an unmanaged C-style array.

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>source</i> , <i>startIndex</i> , <i>destination</i> , or <i>length</i> is null.

Marshal.Copy(System.Int32[], System.Int32, System.IntPtr, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Copy(int32[] source, int32
startIndex, native int destination, int32 length) cil managed

[C#]
public static void Copy (int[] source, int startIndex, IntPtr destination,
int length)
```

Summary

Copies data from a one-dimensional, managed 32-bit signed integer array to an unmanaged memory pointer.

Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

Description

You can use this method to copy a subset of a one-dimensional managed array to an unmanaged C-style array.

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>startIndex</i> or <i>length</i> is null.

18
19

Marshal.Copy(System.Int64[], System.IntPtr, System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Copy(int64[] source, int32
startIndex, native int destination, int32 length) cil managed

[C#]
public static void Copy (long[] source, int startIndex, IntPtr
destination, int length)
```

Summary

Copies data from a one-dimensional, managed 64-bit signed integer array to an unmanaged memory pointer.

Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

Description

You can use this method to copy a subset of a one-dimensional managed array to an unmanaged C-style array.

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>source</i> , <i>startIndex</i> , <i>destination</i> , or <i>length</i> is null.

1 Marshal.Copy(System.IntPtr, System.Byte[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(native int source, uint8[]  
5 destination, int32 startIndex, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (IntPtr source, byte[] destination, int  
8 startIndex, int length)
```

9 Summary

10 Copies data from an unmanaged memory pointer to a managed 8-bit unsigned integer
11 array.

12 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 Unmanaged, C-style arrays do not contain bounds information, which prevents the
16 *startIndex* and *length* parameters from being validated. Thus, the unmanaged data
17 corresponding to the *source* parameter populates the managed array regardless of its
18 usefulness. You must initialize the managed array with the appropriate size before
19 calling this method.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

21

22

1 Marshal.Copy(System.IntPtr, System.Char[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(native int source, char[]  
5 destination, int32 startIndex, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (IntPtr source, char[] destination, int  
8 startIndex, int length)
```

9 Summary

10 Copies data from an unmanaged memory pointer to a managed character array.

11 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

12

13 Description

14 Unmanaged, C-style arrays do not contain bounds information, which prevents the
15 *startIndex* and *length* parameters from being validated. Thus, the unmanaged data
16 corresponding to the *source* parameter populates the managed array regardless of its
17 usefulness. You must initialize the managed array with the appropriate size before
18 calling this method.

19 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

20

21

1 Marshal.Copy(System.IntPtr, 2 System.Double[], System.Int32, 3 System.Int32) Method

```
4 [ILAsm]  
5 .method public hidebysig static void Copy(native int source, float64[]  
6 destination, int32 startIndex, int32 length) cil managed  
  
7 [C#]  
8 public static void Copy (IntPtr source, double[] destination, int  
9 startIndex, int length)
```

10 Summary

11 Copies data from an unmanaged memory pointer to a managed double-precision
12 floating-point number array.

13 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

14 15 Description

16 Unmanaged, C-style arrays do not contain bounds information, which prevents the
17 *startIndex* and *length* parameters from being validated. Thus, the unmanaged data
18 corresponding to the *source* parameter populates the managed array regardless of its
19 usefulness. You must initialize the managed array with the appropriate size before
20 calling this method.

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

22

23

Marshal.Copy(System.IntPtr, System.Int16[], System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig static void Copy(native int source, int16[]
destination, int32 startIndex, int32 length) cil managed

[C#]
public static void Copy (IntPtr source, short[] destination, int
startIndex, int length)
```

Summary

Copies data from an unmanaged memory pointer to a managed 16-bit signed integer array.

Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

Description

Unmanaged, C-style arrays do not contain bounds information, which prevents the *startIndex* and *length* parameters from being validated. Thus, the unmanaged data corresponding to the *source* parameter populates the managed array regardless of its usefulness. You must initialize the managed array with the appropriate size before calling this method.

Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

1 Marshal.Copy(System.IntPtr, System.Int32[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(native int source, int32[]  
5 destination, int32 startIndex, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (IntPtr source, int[] destination, int startIndex,  
8 int length)
```

9 Summary

10 Copies data from an unmanaged memory pointer to a managed 32-bit signed integer
11 array.

12 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 Unmanaged, C-style arrays do not contain bounds information, which prevents the
16 *startIndex* and *length* parameters from being validated. Thus, the unmanaged data
17 corresponding to the *source* parameter populates the managed array regardless of its
18 usefulness. You must initialize the managed array with the appropriate size before
19 calling this method.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

21

22

1 Marshal.Copy(System.IntPtr, System.Int64[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(native int source, int64[]  
5 destination, int32 startIndex, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (IntPtr source, long[] destination, int  
8 startIndex, int length)
```

9 Summary

10 Copies data from an unmanaged memory pointer to a managed 64-bit signed integer
11 array.

12 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 Unmanaged, C-style arrays do not contain bounds information, which prevents the
16 *startIndex* and *length* parameters from being validated. Thus, the unmanaged data
17 corresponding to the *source* parameter populates the managed array regardless of its
18 usefulness. You must initialize the managed array with the appropriate size before
19 calling this method.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

21

22

1 Marshal.Copy(System.IntPtr, 2 System.IntPtr[], System.Int32, 3 System.Int32) Method

```
4 [ILAsm]  
5 .method public hidebysig static void Copy(native int source, native int[]  
6 destination, int32 startIndex, int32 length) cil managed  
  
7 [C#]  
8 public static void Copy (IntPtr source, IntPtr[] destination, int  
9 startIndex, int length)
```

10 Summary

11 Copies data from an unmanaged memory pointer to a managed System.IntPtr array.

12 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index into the array where copying should start.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 Unmanaged, C-style arrays do not contain bounds information, which prevents the
16 *startIndex* and *length* parameters from being validated. Therefore, the unmanaged data
17 that corresponds to the *source* parameter populates the managed array regardless of its
18 usefulness. You must initialize the managed array with the appropriate size before
19 calling the System.Runtime.InteropServices.Marshal.Copy method.

20 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

21

22

1 Marshal.Copy(System.IntPtr, 2 System.Single[], System.Int32, 3 System.Int32) Method

```
4 [ILAsm]  
5 .method public hidebysig static void Copy(native int source, float32[]  
6 destination, int32 startIndex, int32 length) cil managed  
  
7 [C#]  
8 public static void Copy (IntPtr source, float[] destination, int  
9 startIndex, int length)
```

10 Summary

11 Copies data from an unmanaged memory pointer to a managed single-precision
12 floating-point number array.

13 Parameters

Parameter	Description
<i>source</i>	The memory pointer to copy from.
<i>destination</i>	The array to copy to.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>length</i>	The number of array elements to copy.

14 15 Description

16 Unmanaged, C-style arrays do not contain bounds information, which prevents the
17 *startIndex* and *length* parameters from being validated. Thus, the unmanaged data
18 corresponding to the *source* parameter populates the managed array regardless of its
19 usefulness. You must initialize the managed array with the appropriate size before
20 calling this method.

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

22
23

1 Marshal.Copy(System.IntPtr[], System.Int32, 2 System.IntPtr, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void Copy(native int[] source, int32  
5 startIndex, native int destination, int32 length) cil managed  
  
6 [C#]  
7 public static void Copy (IntPtr[] source, int startIndex, IntPtr  
8 destination, int length)
```

9 Summary

10 Copies data from a one-dimensional, managed System.IntPtr array to an unmanaged
11 memory pointer.

12 Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index into the array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

13 14 Description

15 You can use this method to copy a subset of a one-dimensional managed
16 System.IntPtr array to an unmanaged C-style array.

17 Exceptions

Exception	Condition
System.ArgumentNullException	<i>source</i> , <i>destination</i> , <i>startIndex</i> , or <i>length</i> is null.

18
19

1 Marshal.Copy(System.Single[], 2 System.Int32, System.IntPtr, System.Int32) 3 Method

```
4 [ILAsm]  
5 .method public hidebysig static void Copy(float32[] source, int32  
6 startIndex, native int destination, int32 length) cil managed  
  
7 [C#]  
8 public static void Copy (float[] source, int startIndex, IntPtr  
9 destination, int length)
```

10 Summary

11 Copies data from a one-dimensional, managed single-precision floating-point number
12 array to an unmanaged memory pointer.

13 Parameters

Parameter	Description
<i>source</i>	The one-dimensional array to copy from.
<i>startIndex</i>	The zero-based index in the source array where copying should start.
<i>destination</i>	The memory pointer to copy to.
<i>length</i>	The number of array elements to copy.

14 15 Description

16 You can use this method to copy a subset of a one-dimensional managed array to an
17 unmanaged C-style array.

18 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	<i>startIndex</i> and <i>length</i> are not valid.
System.ArgumentNullException	<i>source</i> , <i>startIndex</i> , <i>destination</i> , or <i>length</i> is null.

19

1 Marshal.DestroyStructure(System.IntPtr, 2 System.Type) Method

```
3 [ILAsm]  
4 .method public hidebysig static void DestroyStructure(native int ptr,  
5 class System.Type structuretype) cil managed internalcall  
  
6 [C#]  
7 public static void DestroyStructure (IntPtr ptr, Type structuretype)
```

8 Summary

9 Frees all substructures that the specified unmanaged memory block points to.

10 Parameters

Parameter	Description
<i>ptr</i>	A pointer to an unmanaged block of memory.
<i>structuretype</i>	Type of a formatted class. This provides the layout information necessary to delete the buffer in the <i>ptr</i> parameter.

11

12 Description

13 You can use this method to free reference-type fields, such as strings, of an unmanaged
14 structure. Unlike its fields, a structure can be a value type or a reference type. Value-
15 type structures that contain value-type fields (all blittable) have no references whose
16 memory must be freed. The
17 `System.Runtime.InteropServices.Marshal.StructureToPtr` method uses this
18 method to prevent memory leaks when reusing memory occupied by a structure.

19

20 `System.Runtime.InteropServices.Marshal.DestroyStructure` calls a platform-
21 specific function, which, in turn, frees an allocated string.

22

23 In addition to `System.Runtime.InteropServices.Marshal.DestroyStructure`, the
24 `System.Runtime.InteropServices.Marshal` class provides the
25 `System.Runtime.InteropServices.Marshal.FreeHGlobal` memory deallocation
26 method.

27 Exceptions

Exception	Condition
<code>System.ArgumentException</code>	<i>structureType</i> has an automatic layout. Use sequential or

	explicit instead.
--	-------------------

1

2

1 Marshal.FreeHGlobal(System.IntPtr) Method

```
2 [ILAsm]  
3 .method public hidebysig static void FreeHGlobal(native int hglobal) cil  
4 managed  
5 [C#]  
6 public static void FreeHGlobal (IntPtr hglobal)
```

7 Summary

8 Frees memory previously allocated from the unmanaged memory of the process.

9 Parameters

Parameter	Description
<i>hglobal</i>	The handle returned by the original matching call to <code>System.Runtime.InteropServices.Marshal.AllocHGlobal</code> .

10

11 Description

12 You can use `System.Runtime.InteropServices.Marshal.FreeHGlobal` to free any
13 memory from the global heap allocated by
14 `System.Runtime.InteropServices.Marshal.AllocHGlobal` or
15 `System.Runtime.InteropServices.Marshal.ReAllocHGlobal`. If the *hglobal* parameter
16 is `System.IntPtr.Zero` the method does nothing.

17

18 In addition to `System.Runtime.InteropServices.Marshal.FreeHGlobal`, the
19 `System.Runtime.InteropServices.Marshal` class provides another memory-
20 deallocation method: `System.Runtime.InteropServices.Marshal.DestroyStructure`.

21

1 2 Marshal.GetDelegateForFunctionPointer(System.IntPtr, System.Type) Method 3

```
4 [ILAsm]  
5 .method public hidebysig static class System.Delegate  
6 GetDelegateForFunctionPointer(native int ptr, class System.Type t) cil  
7 managed  
8  
9 [C#]  
10 public static Delegate GetDelegateForFunctionPointer (IntPtr ptr, Type t)
```

10 Summary

11 Converts an unmanaged function pointer to a delegate.

12 Parameters

Parameter	Description
<i>ptr</i>	The unmanaged function pointer to be converted.
<i>t</i>	The type of the delegate to be returned.

13 14 Return Value

15 A delegate instance that can be cast to the appropriate delegate type.

16 Description

17 One can use the
18 `System.Runtime.InteropServices.Marshal.GetDelegateForFunctionPointer` and
19 `System.Runtime.InteropServices.Marshal.GetFunctionPointerForDelegate`
20 methods to marshal delegates in both directions. With
21 `System.Runtime.InteropServices.Marshal.GetDelegateForFunctionPointer`, *ptr* is
22 imported as a `System.IntPtr`. A `System.IntPtr` can be obtained for a managed
23 delegate by calling
24 `System.Runtime.InteropServices.Marshal.GetFunctionPointerForDelegate` and
25 passed as a parameter; it can then be called from inside the unmanaged method. Note
26 that the parameter marshaler can also marshal function pointers to delegates.

27
28 The `System.Runtime.InteropServices.Marshal.GetDelegateForFunctionPointer`
29 method has the following restrictions:

- 30 • Generics are not supported in interop scenarios.
- 31 • You cannot pass an invalid function pointer to this method.

- 1 • You can use this method only for pure unmanaged function pointers.
- 2 • You cannot use this method with function pointers obtained through C++ or from the
- 3 `System.RuntimeMethodHandle.GetFunctionPointer` method.
- 4 • You cannot use this method to create a delegate from a function pointer to another
- 5 managed delegate.

6 **Exceptions**

Exception	Condition
System.ArgumentException	The <i>t</i> parameter is not a delegate or is generic.
System.ArgumentNullException	The <i>ptr</i> parameter is null. -or- The <i>t</i> parameter is null.

7

8

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

Marshal.GetFunctionPointerForDelegate(System.Delegate) Method

```
[ILAsm]  
.method public hidebysig static native int  
GetFunctionPointerForDelegate(class System.Delegate d) cil managed  
  
[C#]  
public static IntPtr GetFunctionPointerForDelegate (Delegate d)
```

Summary

Converts a delegate into a function pointer that is callable from unmanaged code.

Parameters

Parameter	Description
<i>d</i>	The delegate to be passed to unmanaged code.

Return Value

A value that can be passed to unmanaged code, which, in turn, can use it to call the underlying managed delegate.

Description

The delegate *d* is converted to a function pointer that can be passed to unmanaged code using the `__stdcall` calling convention.

You must manually keep the delegate from being collected by the garbage collector from managed code. The garbage collector does not track reference to unmanaged code.

[*Note:* Generics are not supported in interop scenarios.

]

Exceptions

Exception	Condition
System.ArgumentException	The <i>d</i> parameter is a generic type.
System.ArgumentNullException	The <i>d</i> parameter is null.

1

2

1 Marshal.GetLastWin32Error() Method

```
2 [ILAsm]  
3 .method public hidebysig static int32 GetLastWin32Error() cil managed  
4 internalcall  
  
5 [C#]  
6 public static int GetLastWin32Error ()
```

7 Summary

8 Returns the error code returned by the last unmanaged function that was called using
9 platform invoke that has the
10 `System.Runtime.InteropServices.DllImportAttribute.SetLastError` flag set.

11 Return Value

12 The last error code set by a call to the platform-specific error function.

13 Description

14 `System.Runtime.InteropServices.Marshal.GetLastWin32Error` exposes platform-
15 specific error codes. This method exists because it is not safe to make a direct platform
16 invoke call to `GetLastError` to obtain this information. If you want to access this error
17 code, you must call `System.Runtime.InteropServices.Marshal.GetLastWin32Error`
18 instead of writing your own platform invoke definition for `GetLastError` and calling it.

19
20 You can use this method to obtain error codes only if you apply the
21 `System.Runtime.InteropServices.DllImportAttribute` to the method signature and
22 set the `System.Runtime.InteropServices.DllImportAttribute.SetLastError` field to
23 `true`.

24
25 *[Note:* The name of this method reflects the platform-specificity of the original
26 implementation, and is preserved for compatibility. On other platforms this method may
27 be used to get the last platform-specific error.

28
29]

30

1 Marshal.OffsetOf(System.Type, 2 System.String) Method

```
3 [ILAsm]  
4 .method public hidebysig static native int OffsetOf(class System.Type t,  
5 string fieldName) cil managed  
  
6 [C#]  
7 public static IntPtr OffsetOf (Type t, string fieldName)
```

8 Summary

9 Returns the field offset of the unmanaged form of the managed class.

10 Parameters

Parameter	Description
<i>t</i>	A value type or formatted reference type that specifies the managed class. You must apply the <code>System.Runtime.InteropServices.StructLayoutAttribute</code> to the class.
<i>fieldName</i>	The field within the <i>t</i> parameter.

11

12 Return Value

13 The offset, in bytes, for the *fieldName* parameter within the specified class that is
14 declared by platform invoke.

15 Description

16 `System.Runtime.InteropServices.Marshal.OffsetOf` provides the offset in terms of
17 the unmanaged structure layout, which does not necessarily correspond to the offset of
18 the managed structure layout. Marshaling the structure can transform the layout and
19 alter the offset. The *t* parameter can be a value type or a formatted reference type (with
20 either a sequential or explicit layout). You can obtain the size of the entire layout by
21 using the `System.Runtime.InteropServices.Marshal.SizeOf` method.

22 Exceptions

Exception	Condition
System.ArgumentException	The class cannot be exported as a structure.
System.ArgumentNullException	The <i>t</i> parameter is null.

1

2

1 Marshal.PtrToStringAnsi(System.IntPtr)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string PtrToStringAnsi(native int ptr) cil  
5 managed  
6 [C#]  
7 public static string PtrToStringAnsi (IntPtr ptr)
```

8 Summary

9 Copies all characters up to the first null character from an unmanaged ANSI string to a
10 managed `System.String`, and widens each ANSI character to Unicode.

11 Parameters

Parameter	Description
<i>ptr</i>	The address of the first character of the unmanaged string.

12 13 Return Value

14 A managed string that holds a copy of the unmanaged ANSI string. If *ptr* is `null`, the
15 method returns a null string.

16 Description

17 `System.Runtime.InteropServices.Marshal.PtrToStringAnsi` is useful for custom
18 marshaling or when mixing managed and unmanaged code. Because this method
19 creates a copy of the unmanaged string's contents, you must free the original string as
20 appropriate. This method provides the opposite functionality of the
21 `System.Runtime.InteropServices.Marshal.StringToHGlobalAnsi` method.

22

1 Marshal.PtrToStringAnsi(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static string PtrToStringAnsi(native int ptr,  
5 int32 len) cil managed internalcall  
  
6 [C#]  
7 public static string PtrToStringAnsi (IntPtr ptr, int len)
```

8 Summary

9 Allocates a managed `System.String`, copies a specified number of characters from an
10 unmanaged ANSI string into it, and widens each ANSI character to Unicode.

11 Parameters

Parameter	Description
<i>ptr</i>	The address of the first character of the unmanaged string.
<i>len</i>	The byte count of the input string to copy.

12 13 Return Value

14 A managed string that holds a copy of the native ANSI string if the value of the *ptr*
15 parameter is not `null`; otherwise, this method returns `null`.

16 Description

17 `System.Runtime.InteropServices.Marshal.PtrToStringAnsi` is useful for custom
18 marshaling or when mixing managed and unmanaged code. Because this method
19 creates a copy of the unmanaged string's contents, you must free the original string as
20 appropriate. This method provides the opposite functionality of the
21 `System.Runtime.InteropServices.Marshal.StringToHGlobalAnsi` method.

22 Exceptions

Exception	Condition
<code>System.ArgumentException</code>	<i>len</i> is less than zero.

23
24

1 Marshal.PtrToStringAuto(System.IntPtr) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string PtrToStringAuto(native int ptr) cil  
5 managed  
6 [C#]  
7 public static string PtrToStringAuto (IntPtr ptr)
```

8 Summary

9 Allocates a managed `System.String` and copies all characters up to the first null
10 character from a string stored in unmanaged memory into it.

11 Parameters

Parameter	Description
<i>ptr</i>	For Unicode platforms, the address of the first Unicode character.-or- For ANSI platforms, the address of the first ANSI character.

12 13 Return Value

14 A managed string that holds a copy of the unmanaged string if the value of the *ptr*
15 parameter is not `null`; otherwise, this method returns `null`.

16 Description

17 If the current platform is Unicode, each ANSI character is widened to a Unicode
18 character and this method calls
19 `System.Runtime.InteropServices.Marshal.PtrToStringUni`. Otherwise, this method
20 calls `System.Runtime.InteropServices.Marshal.PtrToStringAnsi`.

21
22 `System.Runtime.InteropServices.Marshal.PtrToStringAuto` is useful for custom
23 marshaling or when mixing managed and unmanaged code. Because this method
24 creates a copy of the unmanaged string's contents, you must free the original string as
25 appropriate. `System.Runtime.InteropServices.Marshal.PtrToStringAuto` provides
26 the opposite functionality of the
27 `System.Runtime.InteropServices.Marshal.StringToHGlobalAuto` method.

28

1 Marshal.PtrToStringAuto(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static string PtrToStringAuto(native int ptr,  
5 int32 len) cil managed  
  
6 [C#]  
7 public static string PtrToStringAuto (IntPtr ptr, int len)
```

8 Summary

9 Allocates a managed `System.String` and copies the specified number of characters from
10 a string stored in unmanaged memory into it.

11 Parameters

Parameter	Description
<i>ptr</i>	For Unicode platforms, the address of the first Unicode character.-or- For ANSI plaforms, the address of the first ANSI character.
<i>len</i>	The number of characters to copy.

12

13 Return Value

14 A managed string that holds a copy of the native string if the value of the *ptr* parameter
15 is not null; otherwise, this method returns null.

16 Description

17 On Unicode platforms, this method calls
18 `System.Runtime.InteropServices.Marshal.PtrToStringUni`; on ANSI platforms, it
19 calls `System.Runtime.InteropServices.Marshal.PtrToStringAnsi`. No
20 transformations are done before these methods are called.

21

22 `System.Runtime.InteropServices.Marshal.PtrToStringAuto` is useful for custom
23 marshaling or when mixing managed and unmanaged code. Because this method
24 creates a copy of the unmanaged string's contents, you must free the original string as
25 appropriate. `System.Runtime.InteropServices.Marshal.PtrToStringAuto` provides
26 the opposite functionality of
27 `System.Runtime.InteropServices.Marshal.StringToHGlobalAuto`.

28 Exceptions

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

System.ArgumentException

len is less than zero.

1

2

1 Marshal.PtrToStringUni(System.IntPtr)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string PtrToStringUni(native int ptr) cil  
5 managed  
6 [C#]  
7 public static string PtrToStringUni (IntPtr ptr)
```

8 Summary

9 Allocates a managed `System.String` and copies all characters up to the first null
10 character from an unmanaged Unicode string into it.

11 Parameters

Parameter	Description
<i>ptr</i>	The address of the first character of the unmanaged string.

12 Return Value

14 A managed string that holds a copy of the unmanaged string if the value of the *ptr*
15 parameter is not `null`; otherwise, this method returns `null`.

16 Description

17 `System.Runtime.InteropServices.Marshal.PtrToStringUni` is useful for custom
18 marshaling or for use when mixing managed and unmanaged code. Because this method
19 creates a copy of the unmanaged string's contents, you must free the original string as
20 appropriate. This method provides the opposite functionality of the
21 `System.Runtime.InteropServices.Marshal.StringToHGlobalUni` method.

22

1 Marshal.PtrToStringUni(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static string PtrToStringUni(native int ptr,  
5 int32 len) cil managed internalcall  
  
6 [C#]  
7 public static string PtrToStringUni (IntPtr ptr, int len)
```

8 Summary

9 Allocates a managed `System.String` and copies a specified number of characters from
10 an unmanaged Unicode string into it.

11 Parameters

Parameter	Description
<i>ptr</i>	The address of the first character of the unmanaged string.
<i>len</i>	The number of Unicode characters to copy.

12 13 Return Value

14 A managed string that holds a copy of the unmanaged string if the value of the *ptr*
15 parameter is not `null`; otherwise, this method returns `null`.

16 Description

17 `System.Runtime.InteropServices.Marshal.PtrToStringUni` is useful for custom
18 marshaling or when mixing managed and unmanaged code. Because this method
19 creates a copy of the unmanaged string's contents, you must free the original string as
20 appropriate. This method provides the opposite functionality of the
21 `System.Runtime.InteropServices.Marshal.StringToHGlobalUni` method.

22

1 Marshal.PtrToStructure(System.IntPtr, 2 System.Object) Method

```
3 [ILAsm]  
4 .method public hidebysig static void PtrToStructure(native int ptr, object  
5 structure) cil managed  
  
6 [C#]  
7 public static void PtrToStructure (IntPtr ptr, object structure)
```

8 Summary

9 Marshals data from an unmanaged block of memory to a managed object.

10 Parameters

Parameter	Description
<i>ptr</i>	A pointer to an unmanaged block of memory.
<i>structure</i>	The object to which the data is to be copied. This must be an instance of a formatted class.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.PtrToStructure` is often necessary in
14 interop scenarios when structure parameters are represented as an `System.IntPtr`
15 value. You cannot use this overload method with value types.

16 Exceptions

Exception	Condition
System.ArgumentException	Structure layout is not sequential or explicit. -or- Structure is a boxed value type.

17

18

1 Marshal.PtrToStructure(System.IntPtr, 2 System.Type) Method

```
3 [ILAsm]  
4 .method public hidebysig static object PtrToStructure(native int ptr,  
5 class System.Type structureType) cil managed  
  
6 [C#]  
7 public static object PtrToStructure (IntPtr ptr, Type structureType)
```

8 Summary

9 Marshals data from an unmanaged block of memory to a newly allocated managed
10 object of the specified type.

11 Parameters

Parameter	Description
<i>ptr</i>	A pointer to an unmanaged block of memory.
<i>structureType</i>	The type of object to be created. This object must represent a formatted class or a structure.

12

13 Return Value

14 A managed object containing the data pointed to by the *ptr* parameter.

15 Description

16 `System.Runtime.InteropServices.Marshal.PtrToStructure` is often necessary in
17 interop scenarios invoke when structure parameters are represented as an
18 `System.IntPtr` value. You can pass a value type to this overload method. In this case,
19 the returned object is a boxed instance.

20 Exceptions

Exception	Condition
System.ArgumentException	The <i>structureType</i> parameter layout is not sequential or explicit. -or- The <i>structureType</i> parameter is a generic type.

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

1

2

1 Marshal.ReadByte(System.IntPtr) Method

```
2 [ILAsm]  
3 .method public hidebysig static uint8 ReadByte(native int ptr) cil managed  
4 [C#]  
5 public static byte ReadByte (IntPtr ptr)
```

6 Summary

7 Reads a single byte from unmanaged memory.

8 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory from which to read.

9 Return Value

11 The byte read from unmanaged memory.

12 Description

13 `System.Runtime.InteropServices.Marshal.ReadByte` has an implied offset of 0. This
14 method enables direct interaction with an unmanaged C-style byte array, eliminating the
15 expense of copying an entire unmanaged array (using
16 `System.Runtime.InteropServices.Marshal.Copy`) to a separate managed array before
17 reading its element values.

18 Reading from unaligned memory locations is supported.

20 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format. -or- <i>ptr</i> is null. -or- <i>ptr</i> is invalid.

1

2

1 Marshal.ReadByte(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static uint8 ReadByte(native int ptr, int32 ofs)  
5 cil managed  
  
6 [C#]  
7 public static byte ReadByte (IntPtr ptr, int ofs)
```

8 Summary

9 Reads a single byte at a given offset (or index) from unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory from which to read.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

11

12 Return Value

13 The byte read from unmanaged memory at the given offset.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadByte` enables direct interaction with
16 an unmanaged C-style byte array, eliminating the expense of copying an entire
17 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
18 separate managed array before reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

22

23

1 Marshal.ReadByte(System.Object, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_RU1"  
5 winapi) uint8 ReadByte([in] object marshal(as any) ptr, int32 ofs) cil  
6 managed  
  
7 [C#]  
8 public static byte ReadByte (object ptr, int ofs)
```

9 Summary

10 Reads a single byte at a given offset (or index) from unmanaged memory.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the source object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

12

13 Return Value

14 The byte read from unmanaged memory at the given offset.

15 Description

16 `System.Runtime.InteropServices.Marshal.ReadByte` enables direct interaction with
17 an unmanaged C-style byte array, eliminating the expense of copying an entire
18 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
19 separate managed array before reading its element values.

20

21 Reading from unaligned memory locations is supported.

22 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept

	System.Runtime.InteropServices.ArrayWithOffset parameters.
--	--

1

2

1 Marshal.ReadInt16(System.IntPtr) Method

```
2 [ILAsm]  
3 .method public hidebysig static int16 ReadInt16(native int ptr) cil  
4 managed  
5 [C#]  
6 public static short ReadInt16 (IntPtr ptr)
```

7 Summary

8 Reads a 16-bit signed integer from unmanaged memory.

9 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory from which to read.

10

11 Return Value

12 The 16-bit signed integer read from unmanaged memory.

13 Description

14 `System.Runtime.InteropServices.Marshal.ReadInt16` has an implied offset of 0. This
15 method enables direct interaction with an unmanaged C-style `Int16` array, eliminating
16 the expense of copying an entire unmanaged array (using
17 `System.Runtime.InteropServices.Marshal.Copy`) to a separate managed array before
18 reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-

	<i>ptr</i> is invalid.
--	------------------------

1

2

1 Marshal.ReadInt16(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static int16 ReadInt16(native int ptr, int32 ofs)  
5 cil managed  
  
6 [C#]  
7 public static short ReadInt16 (IntPtr ptr, int ofs)
```

8 Summary

9 Reads a 16-bit signed integer at a given offset from unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory from which to read.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

11

12 Return Value

13 The 16-bit signed integer read from unmanaged memory at the given offset.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadInt16` enables direct interaction with
16 an unmanaged 16-bit signed array, eliminating the expense of copying an entire
17 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
18 separate managed array before reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

22

23

1 Marshal.ReadInt16(System.Object, 2 System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_RI2"  
5 winapi) int16 ReadInt16([in] object marshal(as any) ptr, int32 ofs) cil  
6 managed  
  
7 [C#]  
8 public static short ReadInt16 (object ptr, int ofs)
```

9 Summary

10 Reads a 16-bit signed integer at a given offset from unmanaged memory.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the source object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

12

13 Return Value

14 The 16-bit signed integer read from unmanaged memory at the given offset.

15 Description

16 `System.Runtime.InteropServices.Marshal.ReadInt16` enables direct interaction with
17 an unmanaged 16-bit signed array, eliminating the expense of copying an entire
18 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
19 separate managed array before reading its element values.

20

21 Reading from unaligned memory locations is supported.

22 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept

	System.Runtime.InteropServices.ArrayWithOffset parameters.
--	---

1

2

1 Marshal.ReadInt32(System.IntPtr) Method

```
2 [ILAsm]  
3 .method public hidebysig static int32 ReadInt32(native int ptr) cil  
4 managed  
5 [C#]  
6 public static int ReadInt32 (IntPtr ptr)
```

7 Summary

8 Reads a 32-bit signed integer from unmanaged memory.

9 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory from which to read.

10

11 Return Value

12 The 32-bit signed integer read from unmanaged memory.

13 Description

14 `System.Runtime.InteropServices.Marshal.ReadInt32` has an implied offset of 0. This
15 method enables direct interaction with an unmanaged C-style `Int32` array, eliminating
16 the expense of copying an entire unmanaged array (using
17 `System.Runtime.InteropServices.Marshal.Copy`) to a separate managed array before
18 reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-

	<i>ptr</i> is invalid.
--	------------------------

1

2

1 Marshal.ReadInt32(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 ReadInt32(native int ptr, int32 ofs)  
5 cil managed  
  
6 [C#]  
7 public static int ReadInt32 (IntPtr ptr, int ofs)
```

8 Summary

9 Reads a 32-bit signed integer at a given offset from unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory from which to read.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

11

12 Return Value

13 The 32-bit signed integer read from unmanaged memory.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadInt32` enables direct interaction with
16 an unmanaged 32-bit signed array, eliminating the expense of copying an entire
17 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
18 separate managed array before reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

22

23

1 Marshal.ReadInt32(System.Object, 2 System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_RI4"  
5 winapi) int32 ReadInt32([in] object marshal(as any) ptr, int32 ofs) cil  
6 managed  
  
7 [C#]  
8 public static int ReadInt32 (object ptr, int ofs)
```

9 Summary

10 Reads a 32-bit signed integer at a given offset from unmanaged memory.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the source object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

12

13 Return Value

14 The 32-bit signed integer read from unmanaged memory at the given offset.

15 Description

16 `System.Runtime.InteropServices.Marshal.ReadInt32` enables direct interaction with
17 an unmanaged 32-bit signed array, eliminating the expense of copying an entire
18 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
19 separate managed array before reading its element values.

20

21 Reading from unaligned memory locations is supported.

22 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept

	System.Runtime.InteropServices.ArrayWithOffset parameters.
--	--

1

2

1 Marshal.ReadInt64(System.IntPtr) Method

```
2 [ILAsm]  
3 .method public hidebysig static int64 ReadInt64(native int ptr) cil  
4 managed  
5 [C#]  
6 public static long ReadInt64 (IntPtr ptr)
```

7 Summary

8 Reads a 64-bit signed integer from unmanaged memory.

9 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory from which to read.

10

11 Return Value

12 The 64-bit signed integer read from unmanaged memory.

13 Description

14 `System.Runtime.InteropServices.Marshal.ReadInt64` has an implied offset of 0. This
15 method enables direct interaction with an unmanaged C-style `Int64` array, eliminating
16 the expense of copying an entire unmanaged array (using
17 `System.Runtime.InteropServices.Marshal.Copy`) to a separate managed array before
18 reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-

	<i>ptr</i> is invalid.
--	------------------------

1

2

1 Marshal.ReadInt64(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static int64 ReadInt64(native int ptr, int32 ofs)  
5 cil managed  
  
6 [C#]  
7 public static long ReadInt64 (IntPtr ptr, int ofs)
```

8 Summary

9 Reads a 64-bit signed integer at a given offset from unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory from which to read.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

11

12 Return Value

13 The 64-bit signed integer read from unmanaged memory at the given offset.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadInt64` enables direct interaction with
16 an unmanaged 64-bit signed array, eliminating the expense of copying an entire
17 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
18 separate managed array before reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

22

23

1 Marshal.ReadInt64(System.Object, 2 System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_RI8"  
5 winapi) int64 ReadInt64([in] object marshal(as any) ptr, int32 ofs) cil  
6 managed  
  
7 [C#]  
8 public static long ReadInt64 (object ptr, int ofs)
```

9 Summary

10 Reads a 64-bit signed integer at a given offset from unmanaged memory.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the source object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

12

13 Return Value

14 The 64-bit signed integer read from unmanaged memory at the given offset.

15 Description

16 `System.Runtime.InteropServices.Marshal.ReadInt64` enables direct interaction with
17 an unmanaged 64-bit signed array, eliminating the expense of copying an entire
18 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
19 separate managed array before reading its element values.

20

21 Reading from unaligned memory locations is supported.

22 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept

	System.Runtime.InteropServices.ArrayWithOffset parameters.
--	--

1

2

1 Marshal.ReadIntPtr(System.IntPtr) Method

```
2 [ILAsm]  
3 .method public hidebysig static native int ReadIntPtr(native int ptr) cil  
4 managed  
5 [C#]  
6 public static IntPtr ReadIntPtr (IntPtr ptr)
```

7 Summary

8 Reads a processor native-sized integer from unmanaged memory.

9 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory from which to read.

10

11 Return Value

12 The integer read from unmanaged memory. A 32 bit integer is returned on 32 bit
13 machines and a 64 bit integer is returned on 64 bit machines.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadIntPtr` has an implied offset of 0.
16 This method enables direct interaction with an unmanaged C-style `IntPtr` array,
17 eliminating the expense of copying an entire unmanaged array (using
18 `System.Runtime.InteropServices.Marshal.Copy`) to a separate managed array before
19 reading its element values.

20

21 Reading from unaligned memory locations is supported.

22 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-

	<i>ptr</i> is invalid.
--	------------------------

1

2

1 Marshal.ReadIntPtr(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static native int ReadIntPtr(native int ptr,  
5 int32 ofs) cil managed  
  
6 [C#]  
7 public static IntPtr ReadIntPtr (IntPtr ptr, int ofs)
```

8 Summary

9 Reads a processor native sized integer at a given offset from unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory from which to read.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

11 12 Return Value

13 The integer read from unmanaged memory at the given offset.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadIntPtr` enables direct interaction
16 with an unmanaged C-style `IntPtr` array, eliminating the expense of copying an entire
17 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
18 separate managed array before reading its element values.

19
20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

22

23

1 Marshal.ReadIntPtr(System.Object, 2 System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static native int ReadIntPtr([in] object  
5 marshal(as any) ptr, int32 ofs) cil managed  
  
6 [C#]  
7 public static IntPtr ReadIntPtr (object ptr, int ofs)
```

8 Summary

9 Reads a processor native sized integer from unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the source object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before reading.

11

12 Return Value

13 The integer read from unmanaged memory at the given offset.

14 Description

15 `System.Runtime.InteropServices.Marshal.ReadIntPtr` enables direct interaction
16 with an unmanaged C-style `IntPtr` array, eliminating the expense of copying an entire
17 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
18 separate managed array before reading its element values.

19

20 Reading from unaligned memory locations is supported.

21 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code>

	parameters.
--	-------------

1

2

1 Marshal.ReAllocHGlobal(System.IntPtr, 2 System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static native int ReAllocHGlobal(native int pv,  
5 native int cb) cil managed  
  
6 [C#]  
7 public static IntPtr ReAllocHGlobal (IntPtr pv, IntPtr cb)
```

8 Summary

9 Resizes a block of memory previously allocated with
10 System.Runtime.InteropServices.Marshal.AllocHGlobal.

11 Parameters

Parameter	Description
<i>pv</i>	A pointer to memory allocated with System.Runtime.InteropServices.Marshal.AllocHGlobal.
<i>cb</i>	The new size of the allocated block.

12

13 Return Value

14 A pointer to the reallocated memory. This memory must be released using
15 System.Runtime.InteropServices.Marshal.FreeHGlobal.

16 Description

17 System.Runtime.InteropServices.Marshal.ReAllocHGlobal is the memory allocation
18 API method in the System.Runtime.InteropServices.Marshal class. The returned
19 pointer can differ from the original.

20 Exceptions

Exception	Condition
System.OutOfMemoryException	There is insufficient memory to satisfy the request.

21

22

1 Marshal.SizeOf(System.Object) Method

```
2 [ILAsm]  
3 .method public hidebysig static int32 SizeOf(object structure) cil managed  
4 [C#]  
5 public static int SizeOf (object structure)
```

6 Summary

7 Returns the unmanaged size of an object in bytes.

8 Parameters

Parameter	Description
<i>structure</i>	The object whose size is to be returned.

9 Return Value

11 The size of the specified object in unmanaged code.

12 Description

13 This method accepts an instance of a structure, which can be a reference type or a
14 boxed value type. The layout must be sequential or explicit.

16 The size returned is the size of the unmanaged object. The unmanaged and managed
17 sizes of an object can differ. For character types, the size is affected by the
18 `System.Runtime.InteropServices.CharSet` value applied to that class.

20 You can use the `System.Runtime.InteropServices.Marshal.SizeOf` method to
21 determine how much unmanaged memory to allocate using the
22 `System.Runtime.InteropServices.Marshal.AllocHGlobal` method.

23 Exceptions

Exception	Condition
System.ArgumentNullException	The <i>structure</i> parameter is null.

24
25

1 Marshal.SizeOf(System.Type) Method

```
2 [ILAsm]  
3 .method public hidebysig static int32 SizeOf(class System.Type t) cil  
4 managed  
5 [C#]  
6 public static int SizeOf (Type t)
```

7 Summary

8 Returns the size of an unmanaged type in bytes.

9 Parameters

Parameter	Description
<i>t</i>	The type whose size is to be returned.

10

11 Return Value

12 The size of the specified type in unmanaged code.

13 Description

14 You can use this method when you do not have a structure. The layout must be
15 sequential or explicit.

16
17 The size returned is the size of the unmanaged type. The unmanaged and managed
18 sizes of an object can differ. For character types, the size is affected by the
19 `System.Runtime.InteropServices.CharSet` value applied to that class.

20 Exceptions

Exception	Condition
<code>System.ArgumentException</code>	The <i>t</i> parameter is a generic type.
<code>System.ArgumentNullException</code>	The <i>t</i> parameter is <code>null</code> .

21

22

1 Marshal.StringToHGlobalAnsi(System.String) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static native int StringToHGlobalAnsi(string s)  
5 cil managed  
  
6 [C#]  
7 public static IntPtr StringToHGlobalAnsi (string s)
```

8 Summary

9 Copies the contents of a managed `System.String` into unmanaged memory, converting
10 into ANSI format as it copies.

11 Parameters

Parameter	Description
s	A managed string to be copied.

12 13 Return Value

14 The address, in unmanaged memory, to where `s` was copied, or 0 if `s` is null.

15 Description

16 `System.Runtime.InteropServices.Marshal.StringToHGlobalAnsi` is useful for
17 custom marshaling or when mixing managed and unmanaged code. Because this
18 method allocates the unmanaged memory required for a string, always free the memory
19 by calling `System.Runtime.InteropServices.Marshal.FreeHGlobal`.
20 `System.Runtime.InteropServices.Marshal.StringToHGlobalAnsi` provides the
21 opposite functionality of
22 `System.Runtime.InteropServices.Marshal.PtrToStringAnsi`.

23 Exceptions

Exception	Condition
<code>System.OutOfMemoryException</code>	There is insufficient memory available.
<code>System.ArgumentOutOfRangeException</code>	The <code>s</code> parameter exceeds the maximum length allowed by the operating system.

24
25

1 Marshal.StringToHGlobalAuto(System.String) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static native int StringToHGlobalAuto(string s)  
5 cil managed  
  
6 [C#]  
7 public static IntPtr StringToHGlobalAuto (string s)
```

8 Summary

9 Copies the contents of a managed `System.String` into unmanaged memory, converting
10 into ANSI format if required.

11 Parameters

Parameter	Description
s	A managed string to be copied.

12 13 Return Value

14 The address, in unmanaged memory, to where the string was copied, or 0 if s is null.

15 Description

16 `System.Runtime.InteropServices.Marshal.StringToHGlobalAuto` is useful for
17 custom marshaling or for use when mixing managed and unmanaged code. Because this
18 method allocates the unmanaged memory required for a string, always free the memory
19 by calling `System.Runtime.InteropServices.Marshal.FreeHGlobal`. This method
20 provides the opposite functionality of
21 `System.Runtime.InteropServices.Marshal.PtrToStringAuto`.

22 Exceptions

Exception	Condition
<code>System.OutOfMemoryException</code>	There is insufficient memory available.

23

24

1 Marshal.StringToHGlobalUni(System.String) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static native int StringToHGlobalUni(string s)  
5 cil managed  
  
6 [C#]  
7 public static IntPtr StringToHGlobalUni (string s)
```

8 Summary

9 Copies the contents of a managed `System.String` into unmanaged memory.

10 Parameters

Parameter	Description
s	A managed string to be copied.

11 Return Value

13 The address, in unmanaged memory, to where the s was copied, or 0 if s is null.

14 Description

15 `System.Runtime.InteropServices.Marshal.StringToHGlobalUni` is useful for custom
16 marshaling or for use when mixing managed and unmanaged code. Because this method
17 allocates the unmanaged memory required for a string, always free the memory by
18 calling `System.Runtime.InteropServices.Marshal.FreeHGlobal`. This method
19 provides the opposite functionality of
20 `System.Runtime.InteropServices.Marshal.PtrToStringUni`.

21 Exceptions

Exception	Condition
<code>System.OutOfMemoryException</code>	The method could not allocate enough native heap memory.
<code>System.ArgumentOutOfRangeException</code>	The s parameter exceeds the maximum length allowed by the operating system.

22
23

Marshal.StructureToPtr(System.Object, System.IntPtr, System.Boolean) Method

```
[ILAsm]
.method public hidebysig static void StructureToPtr(object structure,
native int ptr, bool fDeleteOld) cil managed internalcall

[C#]
public static void StructureToPtr (object structure, IntPtr ptr, bool
fDeleteOld)
```

Summary

Marshals data from a managed object to an unmanaged block of memory.

Parameters

Parameter	Description
<i>structure</i>	A managed object holding the data to be marshaled. This object must be an instance of a formatted class.
<i>ptr</i>	A pointer to an unmanaged block of memory, which must be allocated before this method is called.
<i>fDeleteOld</i>	true to have the <code>System.Runtime.InteropServices.Marshal.DestroyStructure</code> method called on the <i>ptr</i> parameter before this method executes. Note that passing <code>false</code> can lead to a memory leak.

Description

`System.Runtime.InteropServices.Marshal.StructureToPtr` copies the contents of *structure* to the pre-allocated block of memory that the *ptr* parameter points to. If the *fDeleteOld* parameter is `true`, the pre-allocated buffer is deleted with the appropriate deletion method on the embedded pointer, but the buffer must contain valid data. This method cleans up every reference field specified in the mirrored managed class.

Suppose that *ptr* points to an unmanaged block of memory. The layout of this block is described by a corresponding managed class, which is specified by *structure*. `System.Runtime.InteropServices.Marshal.StructureToPtr` marshals field values from a structure to a pointer. Suppose the *ptr* block includes a reference field pointing to a string buffer that currently holds "abc", and the corresponding field on the managed side is a string that holds "vwxyz". If you do not specify otherwise, `System.Runtime.InteropServices.Marshal.StructureToPtr` allocates a new unmanaged buffer to hold "vwxyz", and hooks it up to the *ptr* block. This action casts the old buffer "abc" adrift without freeing it (its memory is not released back to the unmanaged heap). The result is an orphan buffer that represents a memory leak in your

1 code. If you set the *fDeleteOld* parameter to true,
2 `System.Runtime.InteropServices.Marshal.StructureToPtr` frees the buffer that
3 holds "abc" before allocating a new buffer for "vwxyz".
4
5 [Note: To pin an existing structure, instead of copying it, use the
6 `System.Runtime.InteropServices.GCHandle` type to create a pinned handle for the
7 structure.
8
9]

10 Exceptions

Exception	Condition
System.ArgumentException	The <i>structure</i> parameter is a generic type.

11

12

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

Marshal.UnsafeAddrOfPinnedArrayElement(System.Array, System.Int32) Method

```
[ILAsm]  
.method public hidebysig static native int  
UnsafeAddrOfPinnedArrayElement(class System.Array arr, int32 index) cil  
managed internalcall  
  
[C#]  
public static IntPtr UnsafeAddrOfPinnedArrayElement (Array arr, int index)
```

Summary

Gets the address of the element at the specified index inside the specified array.

Parameters

Parameter	Description
<i>arr</i>	The array that contains the desired element.
<i>index</i>	The index in the <i>arr</i> parameter of the desired element.

Return Value

The address of *index* inside *arr*.

Description

The array must be pinned using a `System.Runtime.InteropServices.GCHandle` before it is passed to this method. For maximum performance, this method does not validate the array passed to it; this can result in unexpected behavior.

1 Marshal.WriteByte(System.IntPtr, System.Byte) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteByte(native int ptr, uint8 val)  
5 cil managed  
  
6 [C#]  
7 public static void WriteByte (IntPtr ptr, byte val)
```

8 Summary

9 Writes a single byte value to unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory to write to.
<i>val</i>	The value to write.

11 Description

13 System.Runtime.InteropServices.Marshal.WriteByte enables direct interaction with
14 an unmanaged C-style byte array, eliminating the expense of copying an entire
15 unmanaged array (using System.Runtime.InteropServices.Marshal.Copy) to a
16 separate managed array before setting its element values.

17 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format. -or- <i>ptr</i> is null. -or- <i>ptr</i> is invalid.

18
19

1 Marshal.WriteByte(System.IntPtr, 2 System.Int32, System.Byte) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteByte(native int ptr, int32 ofs,  
5 uint8 val) cil managed  
  
6 [C#]  
7 public static void WriteByte (IntPtr ptr, int ofs, byte val)
```

8 Summary

9 Writes a single byte value to unmanaged memory at a specified offset.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory to write to.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteByte` enables direct interaction with
14 an unmanaged C-style byte array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

18

19

1 Marshal.WriteByte(System.Object, 2 System.Int32, System.Byte) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_WU1"  
5 winapi) void WriteByte([in][out] object marshal(as any) ptr, int32 ofs,  
6 uint8 val) cil managed  
  
7 [C#]  
8 public static void WriteByte (object ptr, int ofs, byte val)
```

9 Summary

10 Writes a single byte value to unmanaged memory at a specified offset.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the target object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

12

13 Description

14 `System.Runtime.InteropServices.Marshal.WriteByte` enables direct interaction with
15 an unmanaged C-style byte array, eliminating the expense of copying an entire
16 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
17 separate managed array before setting its element values.

18 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code> parameters.

1

2

1 Marshal.WriteInt16(System.IntPtr, 2 System.Char) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt16(native int ptr, char val)  
5 cil managed  
  
6 [C#]  
7 public static void WriteInt16 (IntPtr ptr, char val)
```

8 Summary

9 Writes a character as a 16-bit integer value to unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory to write to.
<i>val</i>	The value to write.

11

12 Description

13 System.Runtime.InteropServices.Marshal.WriteInt16 enables direct interaction
14 with an unmanaged 16-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using System.Runtime.InteropServices.Marshal.Copy) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-
	<i>ptr</i> is invalid.

1

2

1 Marshal.WriteInt16(System.IntPtr, 2 System.Int16) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt16(native int ptr, int16 val)  
5 cil managed  
  
6 [C#]  
7 public static void WriteInt16 (IntPtr ptr, short val)
```

8 Summary

9 Writes a 16-bit integer value to unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory to write to.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteInt16` enables direct interaction
14 with an unmanaged 16-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format. -or- <i>ptr</i> is null. -or- <i>ptr</i> is invalid.

1

2

1 Marshal.WriteInt16(System.IntPtr, 2 System.Int32, System.Char) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt16(native int ptr, int32 ofs,  
5 char val) cil managed  
  
6 [C#]  
7 public static void WriteInt16 (IntPtr ptr, int ofs, char val)
```

8 Summary

9 Writes a 16-bit signed integer value to unmanaged memory at a specified offset.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in the native heap to write to.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteInt16` enables direct interaction
14 with an unmanaged 16-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

20

21

1 Marshal.WriteInt16(System.IntPtr, 2 System.Int32, System.Int16) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt16(native int ptr, int32 ofs,  
5 int16 val) cil managed  
  
6 [C#]  
7 public static void WriteInt16 (IntPtr ptr, int ofs, short val)
```

8 Summary

9 Writes a 16-bit signed integer value into unmanaged memory at a specified offset.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory to write to.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteInt16` enables direct interaction
14 with an unmanaged 16-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

20

21

1 Marshal.WriteInt16(System.Object, 2 System.Int32, System.Char) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt16([in][out] object ptr,  
5 int32 ofs, char val) cil managed  
  
6 [C#]  
7 public static void WriteInt16 (object ptr, int ofs, char val)
```

8 Summary

9 Writes a 16-bit signed integer value to unmanaged memory at a specified offset.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the target object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteInt16` enables direct interaction
14 with an unmanaged 16-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code>

	parameters.
--	-------------

1

2

1 Marshal.WriteInt16(System.Object, 2 System.Int32, System.Int16) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_WI2"  
5 winapi) void WriteInt16([in][out] object marshal(as any) ptr, int32 ofs,  
6 int16 val) cil managed  
  
7 [C#]  
8 public static void WriteInt16 (object ptr, int ofs, short val)
```

9 Summary

10 Writes a 16-bit signed integer value to unmanaged memory at a specified offset.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the target object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

12

13 Description

14 `System.Runtime.InteropServices.Marshal.WriteInt16` enables direct interaction
15 with an unmanaged 16-bit signed array, eliminating the expense of copying an entire
16 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
17 separate managed array before setting its element values.

18

19 Writing to unaligned memory locations is supported.

20 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code>

	parameters.
--	-------------

1

2

1 Marshal.WriteInt32(System.IntPtr, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt32(native int ptr, int32 val)  
5 cil managed  
  
6 [C#]  
7 public static void WriteInt32 (IntPtr ptr, int val)
```

8 Summary

9 Writes a 32-bit signed integer value to unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory to write to.
<i>val</i>	The value to write.

11

12 Description

13 System.Runtime.InteropServices.Marshal.WriteInt32 enables direct interaction
14 with an unmanaged 32-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using System.Runtime.InteropServices.Marshal.Copy) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-
	<i>ptr</i> is invalid.

1

2

1 Marshal.WriteInt32(System.IntPtr, 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt32(native int ptr, int32 ofs,  
5 int32 val) cil managed  
  
6 [C#]  
7 public static void WriteInt32 (IntPtr ptr, int ofs, int val)
```

8 Summary

9 Writes a 32-bit signed integer value into unmanaged memory at a specified offset.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory to write to.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteInt32` enables direct interaction
14 with an unmanaged 32-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

20

21

1 Marshal.WriteInt32(System.Object, 2 System.IntPtr, System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_WI4"  
5 winapi) void WriteInt32([in][out] object marshal(as any) ptr, int32 ofs,  
6 int32 val) cil managed  
  
7 [C#]  
8 public static void WriteInt32 (object ptr, int ofs, int val)
```

9 Summary

10 Writes a 32-bit signed integer value to unmanaged memory at a specified offset.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the target object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

12

13 Description

14 `System.Runtime.InteropServices.Marshal.WriteInt32` enables direct interaction
15 with an unmanaged 32-bit signed array, eliminating the expense of copying an entire
16 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
17 separate managed array before setting its element values.

18

19 Writing to unaligned memory locations is supported.

20 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code>

	parameters.
--	-------------

1

2

1 Marshal.WriteInt64(System.IntPtr, 2 System.Int64) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt64(native int ptr, int64 val)  
5 cil managed  
  
6 [C#]  
7 public static void WriteInt64 (IntPtr ptr, long val)
```

8 Summary

9 Writes a 64-bit signed integer value to unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory to write to.
<i>val</i>	The value to write.

11

12 Description

13 System.Runtime.InteropServices.Marshal.WriteInt64 enables direct interaction
14 with an unmanaged 64-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using System.Runtime.InteropServices.Marshal.Copy) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-
	<i>ptr</i> is invalid.

1

2

1 Marshal.WriteInt64(System.IntPtr, 2 System.Int32, System.Int64) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteInt64(native int ptr, int32 ofs,  
5 int64 val) cil managed  
  
6 [C#]  
7 public static void WriteInt64 (IntPtr ptr, int ofs, long val)
```

8 Summary

9 Writes a 64-bit signed integer value to unmanaged memory at a specified offset.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory to write.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteInt64` enables direct interaction
14 with an unmanaged 64-bit signed array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

20

21

1 Marshal.WriteInt64(System.Object, 2 System.IntPtr, System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static pinvokeimpl("mscorlib.dll" as "ND_WI8"  
5 winapi) void WriteInt64([in][out] object marshal(as any) ptr, int32 ofs,  
6 int64 val) cil managed  
  
7 [C#]  
8 public static void WriteInt64 (object ptr, int ofs, long val)
```

9 Summary

10 Writes a 64-bit signed integer value to unmanaged memory at a specified offset.

11 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the target object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

12

13 Description

14 `System.Runtime.InteropServices.Marshal.WriteInt64` enables direct interaction
15 with an unmanaged 64-bit signed array, eliminating the expense of copying an entire
16 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
17 separate managed array before setting its element values.

18

19 Writing to unaligned memory locations is supported.

20 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code>

	parameters.
--	-------------

1

2

1 Marshal.IntPtr(System.IntPtr, 2 System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteIntPtr(native int ptr, native  
5 int val) cil managed  
  
6 [C#]  
7 public static void WriteIntPtr (IntPtr ptr, IntPtr val)
```

8 Summary

9 Writes a processor native sized integer value into unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The address in unmanaged memory to write to.
<i>val</i>	The value to write.

11

12 Description

13 System.Runtime.InteropServices.Marshal.WriteIntPtr enables direct interaction
14 with an unmanaged C-style IntPtr array, eliminating the expense of copying an entire
15 unmanaged array (using System.Runtime.InteropServices.Marshal.Copy) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	<i>ptr</i> is not a recognized format.
	-or-
	<i>ptr</i> is null.
	-or-
	<i>ptr</i> is invalid.

1

2

Marshal.WriteIntPtr(System.IntPtr, System.Int32, System.IntPtr) Method

```
[ILAsm]  
.method public hidebysig static void WriteIntPtr(native int ptr, int32  
ofs, native int val) cil managed  
  
[C#]  
public static void WriteIntPtr (IntPtr ptr, int ofs, IntPtr val)
```

Summary

Writes a processor native-sized integer value to unmanaged memory at a specified offset.

Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory to write to.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

Description

This method writes a 32 bit integer on 32 bit systems, and a 64 bit integer on 64 bit systems.

`System.Runtime.InteropServices.Marshal.WriteIntPtr` enables direct interaction with an unmanaged C-style `IntPtr` array, eliminating the expense of copying an entire unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a separate managed array before setting its element values.

Writing to unaligned memory locations is supported.

Exceptions

Exception	Condition
<code>System.AccessViolationException</code>	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.

1 Marshal.WriteIntPtr(System.Object, 2 System.IntPtr, System.IntPtr) Method

```
3 [ILAsm]  
4 .method public hidebysig static void WriteIntPtr([in][out] object  
5 marshal(as any) ptr, int32 ofs, native int val) cil managed  
  
6 [C#]  
7 public static void WriteIntPtr (object ptr, int ofs, IntPtr val)
```

8 Summary

9 Writes a processor native sized integer value to unmanaged memory.

10 Parameters

Parameter	Description
<i>ptr</i>	The base address in unmanaged memory of the target object.
<i>ofs</i>	An additional byte offset, which is added to the <i>ptr</i> parameter before writing.
<i>val</i>	The value to write.

11

12 Description

13 `System.Runtime.InteropServices.Marshal.WriteIntPtr` enables direct interaction
14 with an unmanaged C-style byte array, eliminating the expense of copying an entire
15 unmanaged array (using `System.Runtime.InteropServices.Marshal.Copy`) to a
16 separate managed array before setting its element values.

17

18 Writing to unaligned memory locations is supported.

19 Exceptions

Exception	Condition
System.AccessViolationException	Base address (<i>ptr</i>) plus offset byte (<i>ofs</i>) produces a null or invalid address.
System.ArgumentException	<i>ptr</i> is an <code>System.Runtime.InteropServices.ArrayWithOffset</code> object. This method does not accept <code>System.Runtime.InteropServices.ArrayWithOffset</code>

	parameters.
--	-------------

1

2