

# LinksPlatform's Platform.Unsafe Class Library

## 1.1 ./csharp/Platform.Unsafe/ByteArrayExtensions.cs

```
1 using Platform.Exceptions;
2 using Platform.Collections;
3 using System.Runtime.CompilerServices;
4 using static System.Runtime.CompilerServices.Unsafe;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Unsafe
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the byte array extensions.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public unsafe static class ByteArrayExtensions
17     {
18         /// <summary>
19         /// <para>
20         /// Returns the structure using the specified bytes.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <typeparam name="TStruct">
25         /// <para>The struct.</para>
26         /// <para></para>
27         /// </typeparam>
28         /// <param name="bytes">
29         /// <para>The bytes.</para>
30         /// <para></para>
31         /// </param>
32         /// <returns>
33         /// <para>The structure.</para>
34         /// <para></para>
35         /// </returns>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static TStruct ToStructure<TStruct>(this byte[] bytes)
38             where TStruct : struct
39         {
40             Ensure.OnDebug.ArgumentNotEmpty(bytes, nameof(bytes));
41             Ensure.OnDebug.ArgumentMeetsCriteria(bytes, HasSameSizeAs<TStruct>, nameof(bytes),
42                 ↪ "Bytes array should be the same length as struct size.");
43             TStruct structure = default;
44             fixed (byte* pointer = bytes)
45             {
46                 Copy(ref structure, pointer);
47             }
48             return structure;
49         }
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         private static bool HasSameSizeAs<TStruct>(byte[] array) where TStruct : struct =>
52             ↪ array.Length == Structure<TStruct>.Size;
53     }
54 }
```

## 1.2 ./csharp/Platform.Unsafe/IntPtrExtensions.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
3 using static System.Runtime.CompilerServices.Unsafe;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Unsafe
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the int ptr extensions.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public unsafe static class IntPtrExtensions
16     {
17         /// <summary>
18         /// <para>
19         /// Writes the element value using the specified pointer.
20         /// </para>
21         /// <para></para>
```

```

22     /// </summary>
23     /// <typeparam name="TValue">
24     /// <para>The value.</para>
25     /// <para></para>
26     /// </typeparam>
27     /// <param name="pointer">
28     /// <para>The pointer.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="index">
32     /// <para>The index.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="value">
36     /// <para>The value.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public static void WriteElementValue<TValue>(this IntPtr pointer, long index, TValue
        ↳ value) => Write((byte*)pointer + (SizeOf<TValue>() * index), value);
41
42     /// <summary>
43     /// <para>
44     /// Reads the element value using the specified pointer.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <typeparam name="TValue">
49     /// <para>The value.</para>
50     /// <para></para>
51     /// </typeparam>
52     /// <param name="pointer">
53     /// <para>The pointer.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="index">
57     /// <para>The index.</para>
58     /// <para></para>
59     /// </param>
60     /// <returns>
61     /// <para>The value</para>
62     /// <para></para>
63     /// </returns>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public static TValue ReadElementValue<TValue>(this IntPtr pointer, long index) =>
        ↳ Read<TValue>((byte*)pointer + (SizeOf<TValue>() * index));
66 }
67 }

```

### 1.3 ./csharp/Platform.Unsafe/MemoryBlock.cs

```

1  using System;
2  using System.Collections.Concurrent;
3  using System.Runtime.CompilerServices;
4  using System.Threading.Tasks;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Unsafe
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the memory block.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static unsafe class MemoryBlock
18     {
19         /// <summary>
20         /// <para>
21         /// Zeroes the pointer.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="pointer">
26         /// <para>The pointer.</para>
27         /// <para></para>
28         /// </param>

```

```

29     /// <param name="capacity">
30     /// <para>The capacity.</para>
31     /// <para></para>
32     /// </param>
33     [MethodImpl(MethodImplOptions.AggressiveInlining)]
34     public static void Zero(void* pointer, long capacity)
35     {
36         // A way to prevent wasting resources due to Hyper-Threading.
37         var threads = Environment.ProcessorCount / 2;
38         if (threads <= 1)
39         {
40             ZeroBlock(pointer, 0, capacity);
41         }
42         else
43         {
44             // Using 2 threads because two-channel memory architecture is the most available
45             // type.
46             // CPUs mostly just wait for memory here.
47             threads = 2;
48             Parallel.ForEach(Partitioner.Create(OL, capacity), new ParallelOptions {
49                 MaxDegreeOfParallelism = threads }, range => ZeroBlock(pointer, range.Item1,
50                 range.Item2));
51         }
52     }
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     private static void ZeroBlock(void* pointer, long from, long to)
55     {
56         var offset = (byte*)pointer + from;
57         var length = to - from;
58         var uintMaxValue = uint.MaxValue;
59         while (length > uintMaxValue)
60         {
61             InitBlock(offset, 0, uintMaxValue);
62             length -= uintMaxValue;
63             offset += uintMaxValue;
64         }
65         InitBlock(offset, 0, unchecked((uint)length));
66     }
67 }

```

#### 1.4 ./csharp/Platform.Unsafe/Structure.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using System.Runtime.InteropServices;
4 using static System.Runtime.CompilerServices.Unsafe;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Unsafe
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the structure.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public static class Structure<TStruct>
17         where TStruct : struct
18     {
19         /// <summary>
20         /// <para>
21         /// Returns the size of an unmanaged type in bytes.
22         /// This property do this without throwing exceptions for generic types as <see
23         // cref="Marshal.SizeOf{T}()" /> and <see cref="Marshal.SizeOf(Type)" /> do.
24         /// </para>
25         /// <para>
26         /// Возвращает размер неуправляемого типа в байтах.
27         /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
28         // это делают <see cref="Marshal.SizeOf{T}()" /> и <see cref="Marshal.SizeOf(Type)" />.
29         /// </para>
30         /// </summary>
31         public static int Size
32         {
33             [MethodImpl(MethodImplOptions.AggressiveInlining)]
34             get;
35         } = SizeOf<TStruct>();
36     }
37 }

```

### 1.5 ./csharp/Platform.Unsafe/StructureExtensions.cs

```
1 using System.Runtime.CompilerServices;
2 using static System.Runtime.CompilerServices.Unsafe;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Unsafe
7 {
8     /// <summary>
9     /// <para>Represents a set of extension methods for structs.</para>
10    /// <para>Представляет набор методов расширения для структур.</para>
11    /// </summary>
12    public unsafe static class StructureExtensions
13    {
14        /// <summary>
15        /// <para>this process does something</para>
16        /// <para>этот процесс что-то делает</para>
17        /// </summary>
18        [MethodImpl(MethodImplOptions.AggressiveInlining)]
19        public static byte[] ToBytes<TStruct>(this ref TStruct obj)
20            where TStruct : struct
21        {
22            var bytes = new byte[Structure<TStruct>.Size];
23            fixed (byte* pointer = bytes)
24            {
25                Copy(pointer, ref obj);
26            }
27            return bytes;
28        }
29    }
30 }
```

### 1.6 ./csharp/Platform.Unsafe.Tests/IntPtrExtensionsTests.cs

```
1 using System;
2 using System.Runtime.InteropServices;
3 using Xunit;
4 using static System.Runtime.CompilerServices.Unsafe;
5
6 namespace Platform.Unsafe.Tests
7 {
8     public unsafe class IntPtrExtensionsTests
9     {
10         [Fact]
11         public void ReadAndWriteOperationsForPointerValuesUnsafeClassMethodsTest()
12         {
13             void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong));
14             Write(pointer, 42UL);
15             Assert.Equal(42UL, Read<ulong>(pointer));
16             Marshal.FreeHGlobal((IntPtr)pointer);
17         }
18
19         [Fact]
20         public void ElementOffsetOperationsForPointerValuesTest()
21         {
22             void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong) * 10);
23             ulong result = (ulong)Add<ulong>(pointer, 5);
24             Assert.Equal(5UL * 8UL, result - (ulong)pointer);
25             Marshal.FreeHGlobal((IntPtr)pointer);
26         }
27     }
28 }
```

### 1.7 ./csharp/Platform.Unsafe.Tests/SizeOfTests.cs

```
1 using System.Runtime.InteropServices;
2 using Xunit;
3
4 namespace Platform.Unsafe.Tests
5 {
6     public static class SizeOfTests
7     {
8         public struct X<T>
9         {
10             public readonly T F1;
11             public readonly T F2;
12         }
13
14         [Fact]
15         public static void UnsafeClassSizeOfTest()
16         {
17         }
```

```

17         var size = System.Runtime.CompilerServices.Unsafe.SizeOf<X<int>>>();
18         Assert.Equal(8, size);
19     }
20
21     [Fact]
22     public static void MarshalSizeOfTest()
23     {
24         var size = Marshal.SizeOf(default(X<int>));
25         Assert.Equal(8, size);
26     }
27
28     [Fact]
29     public static void StructurePropertyTest()
30     {
31         var size = Structure<X<int>>.Size;
32         Assert.Equal(8, size);
33     }
34 }
35

```

## 1.8 ./csharp/Platform.Unsafe.Tests/StructAndBytesConversionTests.cs

```

1  using Xunit;
2
3  namespace Platform.Unsafe.Tests
4  {
5      public static class StructAndBytesConversionTests
6      {
7          [Fact]
8          public static void StructToBytesTest()
9          {
10             ulong source = ulong.MaxValue;
11             var result = source.ToBytes();
12             for (int i = 0; i < result.Length; i++)
13             {
14                 Assert.Equal(byte.MaxValue, result[i]);
15             }
16         }
17
18         [Fact]
19         public static void BytesToStructTest()
20         {
21             byte[] bytes = new[] { byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue,
22                                     ↳ byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue };
23             ulong result = bytes.ToStructure<ulong>();
24             Assert.Equal(ulong.MaxValue, result);
25         }
26     }
27

```

## 1.9 ./csharp/Platform.Unsafe.Tests/ZeroMemoryTests.cs

```

1  using Xunit;
2
3  namespace Platform.Unsafe.Tests
4  {
5      public static unsafe class ZeroMemoryTests
6      {
7          [Fact]
8          public static void ZeroMemoryTest()
9          {
10             var bytes = new byte[1024];
11             for (int i = 0; i < bytes.Length; i++)
12             {
13                 bytes[i] = unchecked((byte)i);
14             }
15             fixed (byte* pointer = bytes)
16             {
17                 MemoryBlock.Zero(pointer, bytes.Length);
18             }
19             for (int i = 0; i < bytes.Length; i++)
20             {
21                 Assert.Equal(0, bytes[i]);
22             }
23         }
24     }
25

```

## Index

- ./csharp/Platform.Unsafe.Tests/IntPtrExtensionsTests.cs, 4
- ./csharp/Platform.Unsafe.Tests/SizeOfTests.cs, 4
- ./csharp/Platform.Unsafe.Tests/StructAndBytesConversionTests.cs, 5
- ./csharp/Platform.Unsafe.Tests/ZeroMemoryTests.cs, 5
- ./csharp/Platform.Unsafe/ByteArrayExtensions.cs, 1
- ./csharp/Platform.Unsafe/IntPtrExtensions.cs, 1
- ./csharp/Platform.Unsafe/MemoryBlock.cs, 2
- ./csharp/Platform.Unsafe/Structure.cs, 3
- ./csharp/Platform.Unsafe/StructureExtensions.cs, 4