

## 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     public unsafe static class ByteArrayExtensions
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static TStruct ToStructure<TStruct>(this byte[] bytes)
14             where TStruct : struct
15         {
16             Ensure.OnDebug.ArgumentNotEmpty(bytes, nameof(bytes));
17             Ensure.OnDebug.ArgumentMeetsCriteria(bytes, HasSameSizeAs<TStruct>, nameof(bytes),
18                 ↪ "Bytes array should be the same length as struct size.");
19             TStruct structure = default;
20             fixed (byte* pointer = bytes)
21             {
22                 Copy(ref structure, pointer);
23             }
24             return structure;
25         }
26
27         private static bool HasSameSizeAs<TStruct>(byte[] array) where TStruct : struct =>
28             ↪ array.Length == Structure<TStruct>.Size;
29     }
30 }
```

### 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     public unsafe static class IntPtrExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static void WriteElementValue<TValue>(this IntPtr pointer, long index, TValue
13             ↪ value) => Write((byte*)pointer + (SizeOf<TValue>() * index), value);
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static TValue ReadElementValue<TValue>(this IntPtr pointer, long index) =>
17             ↪ Read<TValue>((byte*)pointer + (SizeOf<TValue>() * index));
18     }
19 }
```

### 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     public static unsafe class MemoryBlock
12     {
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public static void Zero(void* pointer, long capacity)
15         {
16             // A way to prevent wasting resources due to Hyper-Threading.
17             var threads = Environment.ProcessorCount / 2;
18             if (threads <= 1)
19             {
20                 ZeroBlock(pointer, 0, capacity);
21             }
22             else
23             {
24                 // Using 2 threads because two-channel memory architecture is the most available
25                 ↪ type.
26             }
27         }
28     }
29 }
```

```

25         // CPUs mostly just wait for memory here.
26         threads = 2;
27         Parallel.ForEach(Partitioner.Create(OL, capacity), new ParallelOptions {
            ↪ MaxDegreeOfParallelism = threads }, range => ZeroBlock(pointer, range.Item1,
            ↪ range.Item2));
28     }
29 }
30
31 [MethodImpl(MethodImplOptions.AggressiveInlining)]
32 private static void ZeroBlock(void* pointer, long from, long to)
33 {
34     var offset = (byte*)pointer + from;
35     var length = to - from;
36     var uintMaxValue = uint.MaxValue;
37     while (length > uintMaxValue)
38     {
39         InitBlock(offset, 0, uintMaxValue);
40         length -= uintMaxValue;
41         offset += uintMaxValue;
42     }
43     InitBlock(offset, 0, unchecked((uint)length));
44 }
45 }
46 }

```

#### 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     public static class Structure<TStruct>
11         where TStruct : struct
12     {
13         /// <summary>
14         /// <para>
15         /// Returns the size of an unmanaged type in bytes.
16         /// This property do this without throwing exceptions for generic types as <see
            ↪ cref="Marshal.SizeOf{T}()" /> and <see cref="Marshal.SizeOf(Type)" /> do.
17         /// </para>
18         /// <para>
19         /// Возвращает размер неуправляемого типа в байтах.
20         /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
            ↪ это делают <see cref="Marshal.SizeOf{T}()" /> и <see cref="Marshal.SizeOf(Type)" />.
21         /// </para>
22         /// </summary>
23         public static int Size
24         {
25             [MethodImpl(MethodImplOptions.AggressiveInlining)]
26             get;
27             } = SizeOf<TStruct>();
28     }
29 }

```

#### 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     public unsafe static class StructureExtensions
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public static byte[] ToBytes<TStruct>(this ref TStruct obj)
12             where TStruct : struct
13         {
14             var bytes = new byte[Structure<TStruct>.Size];
15             fixed (byte* pointer = bytes)
16             {
17                 Copy(pointer, ref obj);
18             }
19             return bytes;
20         }
21     }
22 }

```

```
21     }
22 }
```

### 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             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 }

```

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