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

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

→ Read<TValue>((byte*)pointer + (SizeOf<TValue>() * index));
        }
66
   }
    ./csharp/Platform.Unsafe/MemoryBlock.cs
   using System;
   using System.Collections.Concurrent;
   using System.Runtime.CompilerServices; using System.Threading.Tasks;
3
4
   using static System.Runtime.CompilerServices.Unsafe;
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Unsafe
9
10
        /// <summary>
11
        /// <para>
12
        /// Represents the memory block.
13
        /// </para>
14
        /// <para></para>
        /// </summary>
16
        public static unsafe class MemoryBlock
17
            /// <summary>
19
            /// <para>
20
            /// Zeroes the pointer.
            /// </para>
22
            /// <para></para>
23
            /// </summary>
24
            /// <param name="pointer">
            /// <para>The pointer.</para>
26
            /// <para></para>
27
            /// </param>
```

```
/// <param name="capacity">
29
            /// <para>The capacity.</para>
            /// <para></para>
31
            /// </param>
32
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
           public static void Zero(void* pointer, long capacity)
34
35
                // A way to prevent wasting resources due to Hyper-Threading.
36
                var threads = Environment.ProcessorCount / 2;
                if (threads <= 1)</pre>
38
                {
39
                    ZeroBlock(pointer, 0, capacity);
40
                }
41
                else
42
                {
43
                    // Using 2 threads because two-channel memory architecture is the most available
44
                        type.
                    // CPUs mostly just wait for memory here.
45
                    threads = 2;
46
                    Parallel.ForEach(Partitioner.Create(OL, capacity), new ParallelOptions {
47
                     __ MaxDegreeOfParallelism = threads }, range => ZeroBlock(pointer, range.Item1,

    range.Item2));
                }
48
49
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            private static void ZeroBlock(void* pointer, long from, long to)
51
52
                var offset = (byte*)pointer + from;
53
                var length = to - from;
54
                var uintMaxValue = uint.MaxValue;
                while (length > uintMaxValue)
56
57
58
                    InitBlock(offset, 0, uintMaxValue);
                    length -= uintMaxValue;
59
                    offset += uintMaxValue;
60
61
                InitBlock(offset, 0, unchecked((uint)length));
62
            }
       }
64
65
    ./csharp/Platform.Unsafe/Structure.cs
   using System;
   using System.Runtime.CompilerServices; using System.Runtime.InteropServices;
2
   using static System.Runtime.CompilerServices.Unsafe;
4
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
   namespace Platform. Unsafe
9
   {
        /// <summary>
10
       /// <para>
11
        /// Represents the structure.
12
       /// </para>
13
       /// <para></para>
14
       /// </summary>
15
       public static class Structure<TStruct>
16
            where TStruct : struct
17
18
            /// <summary>
19
            /// <para>
            /// Returns the size of an unmanaged type in bytes.
21
            /// This property do this without throwing exceptions for generic types as <see
22
            /// </para>
23
            /// <para>
24
            /// Возвращает размер неуправляемого типа в байтах.
25
            /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
26
               это делают <see cref="Marshal.SizeOf{T}()"/> и <see cref="Marshal.SizeOf(Type)"/>.
            /// </para>
            /// </summary>
           public static int Size
29
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
31
            } = SizeOf<TStruct>();
       }
34
   }
35
```

```
./csharp/Platform.Unsafe/StructureExtensions.cs
   using System.Runtime.CompilerServices;
   using static System.Runtime.CompilerServices.Unsafe;
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
   namespace Platform. Unsafe
        /// <summary>
        /// <para>Represents a set of extension methods for strucrs.</para>
9
        /// <para>Представляет набор методов расширения для структур.</para>
10
        /// </summary>
11
       public unsafe static class StructureExtensions
12
13
            /// <summary>
14
            /// <para>this process does something</para>
15
            /// <para>этот процесс что-то делает</para>
16
            /// </summary>
17
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
18
            public static byte[] ToBytes<TStruct>(this ref TStruct obj)
19
                where TStruct : struct
20
21
                var bytes = new byte[Structure<TStruct>.Size];
22
                fixed (byte* pointer = bytes)
23
24
                    Copy(pointer, ref obj);
                }
                return bytes;
27
            }
       }
29
30
     ./csharp/Platform.Unsafe.Tests/IntPtrExtensionsTests.cs
1.6
   using System;
   using System.Runtime.InteropServices;
   using Xunit;
3
   using static System.Runtime.CompilerServices.Unsafe;
   namespace Platform. Unsafe. Tests
6
7
        public unsafe class IntPtrExtensionsTests
8
9
            [Fact]
10
            public void ReadAndWriteOperationsForPointerValuesUnsafeClassMethodsTest()
12
                void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong));
13
                Write(pointer, 42UL);
14
                Assert.Equal(42UL, Read<ulong>(pointer));
15
                Marshal.FreeHGlobal((IntPtr)pointer);
16
            }
18
            [Fact]
19
            public void ElementOffsetOperationsForPointerValuesTest()
20
21
                void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong) * 10);
22
                ulong result = (ulong)Add<ulong>(pointer, 5);
                Assert.Equal(5UL * 8UL, result - (ulong)pointer);
24
                Marshal.FreeHGlobal((IntPtr)pointer);
25
            }
26
       }
27
28
     ./csharp/Platform.Unsafe.Tests/SizeOfTests.cs
   using System.Runtime.InteropServices;
   using Xunit;
2
3
   namespace Platform. Unsafe. Tests
4
5
        public static class SizeOfTests
6
            public struct X<T>
                public readonly T F1;
10
                public readonly T F2;
11
12
13
            [Fact]
14
            public static void UnsafeClassSizeOfTest()
15
```

```
var size = System.Runtime.CompilerServices.Unsafe.SizeOf<X<int>>();
17
                 Assert.Equal(8, size);
18
            }
19
            [Fact]
21
            public static void MarshalSizeOfTest()
22
23
                 var size = Marshal.SizeOf(default(X<int>));
^{24}
                 Assert.Equal(8, size);
25
            }
26
27
            [Fact]
28
29
            public static void StructurePropertyTest()
30
                 var size = Structure<X<int>>.Size;
31
                 Assert.Equal(8, size);
            }
33
        }
34
   }
35
    ./csharp/Platform.Unsafe.Tests/StructAndBytesConversionTests.cs
1.8
   using Xunit;
   namespace Platform. Unsafe. Tests
        public static class StructAndBytesConversionTests
5
6
            [Fact]
            public static void StructToBytesTest()
                 ulong source = ulong.MaxValue;
10
                 var result = source.ToBytes();
11
12
                 for (int i = 0; i < result.Length; i++)</pre>
13
                     Assert.Equal(byte.MaxValue, result[i]);
14
                 }
15
            }
16
17
            [Fact]
18
            public static void BytesToStructTest()
19
20
                 byte[] bytes = new[] { byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue,
                 → byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue };
                 ulong result = bytes.ToStructure<ulong>();
22
                 Assert.Equal(ulong.MaxValue, result);
23
            }
^{24}
        }
^{25}
26
     ./csharp/Platform.Unsafe.Tests/ZeroMemoryTests.cs
1.9
   using Xunit;
   namespace Platform. Unsafe. Tests
3
4
5
        public static unsafe class ZeroMemoryTests
6
            [Fact]
            public static void ZeroMemoryTest()
9
                 var bytes = new byte[1024];
10
                 for (int i = 0; i < bytes.Length; i++)</pre>
11
12
                     bytes[i] = unchecked((byte)i);
13
14
                 fixed (byte* pointer = bytes)
16
                     MemoryBlock.Zero(pointer, bytes.Length);
17
18
                 for (int i = 0; i < bytes.Length; i++)</pre>
19
20
                     Assert.Equal(0, bytes[i]);
21
22
            }
23
        }
24
   }
```

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