

## LinksPlatform's Platform.Unsafe Class Library

### ./Platform.Unsafe/ByteArrayExtensions.cs

```
1 using Platform.Exceptions;
2 using Platform.Collections;
3 using System.Runtime.CompilerServices;
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 ByteArrayExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static TStruct ToStructure<TStruct>(this byte[] bytes)
13             where TStruct : struct
14         {
15             Ensure.OnDebug.ArgumentNotEmpty(bytes, nameof(bytes));
16             var structureSize = System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
17             Ensure.OnDebug.ArgumentMeetsCriteria(bytes, array => array.Length == structureSize,
18                 ↳ nameof(bytes), "Bytes array should be the same length as struct size.");
19             TStruct structure = default;
20             fixed (byte* pointer = bytes)
21             {
22                 System.Runtime.CompilerServices.Unsafe.Copy(ref structure, pointer);
23             }
24             return structure;
25         }
26     }
27 }
```

### ./Platform.Unsafe/IntPtr.cs

```
1 using System;
2 using System.Reflection;
3 using System.Runtime.InteropServices;
4 using Platform.Reflection;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Unsafe
9 {
10     /// <remarks>
11     /// Please use System.Runtime.CompilerServices.Unsafe instead.
12     /// </remarks>
13     public static class IntPtr<T>
14     {
15         public static readonly Func<IntPtr, T> GetValue;
16         public static readonly Action<IntPtr, T> SetValue;
17
18         static IntPtr()
19         {
20             GetValue = CompileGetValueDelegate();
21             SetValue = CompileSetValueDelegate();
22         }
23
24         static private Func<IntPtr, T> CompileGetValueDelegate()
25         {
26             return DelegateHelpers.Compile<Func<IntPtr, T>>(emitter =>
27             {
28                 if (NumericType<T>.IsNumeric)
29                 {
30                     emitter.LoadArgument(0);
31                     emitter.LoadIndirect<T>();
32                     emitter.Return();
33                 }
34                 else
35                 {
36                     emitter.LoadArguments(0);
37                     emitter.Call(typeof(Marshal).GetGenericMethod(nameof(Marshal.PtrToStructure),
38                         ↳ Types<T>.Array, Types<IntPtr, Type, bool>.Array));
39                     emitter.Return();
40                 }
41             });
42         }
43
44         static private Action<IntPtr, T> CompileSetValueDelegate()
45         {
46             return DelegateHelpers.Compile<Action<IntPtr, T>>(emitter =>
47             {
48                 if (NumericType<T>.IsNumeric)
```

```

48         {
49             emitter.LoadArguments(0, 1);
50             emitter.StoreIndirect<T>();
51             emitter.Return();
52         }
53         else
54         {
55             emitter.LoadArguments(0, 1);
56             emitter.LoadConstant(true);
57             emitter.Call(typeof(Marshal).GetTypeInfo().GetMethod(nameof(Marshal.Structure_
                ↳ IntPtr), Types<object, IntPtr,
                ↳ bool>.Array));
58             emitter.Return();
59         }
60     });
61 }
62 }
63 }

```

#### ./Platform.Unsafe/IntPtrExtensions.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Numbers;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Unsafe
8  {
9      /// <remarks>
10     /// Please use System.Runtime.CompilerServices.Unsafe instead.
11     /// </remarks>
12     public static class IntPtrExtensions
13     {
14         [Obsolete("GetValue method is deprecated, please use
        ↳ System.Runtime.CompilerServices.Unsafe.Read method instead.")]
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static TElement GetValue<TElement>(this IntPtr pointer) =>
            ↳ IntPtr<TElement>.GetValue(pointer);
17
18         [Obsolete("SetValue method is deprecated, please use
        ↳ System.Runtime.CompilerServices.Unsafe.Write method instead.")]
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public static void SetValue<TElement>(this IntPtr pointer, TElement value) =>
            ↳ IntPtr<TElement>.SetValue(pointer, value);
21
22         [Obsolete("GetElement method is deprecated, please use
        ↳ System.Runtime.CompilerServices.Unsafe.Add method instead.")]
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static IntPtr GetElement(this IntPtr pointer, int elementSize, int index) =>
            ↳ pointer + (elementSize * index);
25
26         [Obsolete("GetElement method is deprecated, please use
        ↳ System.Runtime.CompilerServices.Unsafe.Add method instead.")]
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static unsafe IntPtr GetElement(this IntPtr pointer, long elementSize, long
            ↳ index) => new IntPtr((byte*)pointer.ToPointer() + (elementSize * index));
29
30         [Obsolete("GetElement method is deprecated, please use
        ↳ System.Runtime.CompilerServices.Unsafe.Add method instead.")]
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public static IntPtr GetElement<TIndex>(this IntPtr pointer, int elementSize, TIndex
            ↳ index) => pointer.GetElement((long)elementSize, (Integer)(Integer<TIndex>)index);
33     }
34 }

```

#### ./Platform.Unsafe/MemoryBlock.cs

```

1  using System.Collections.Concurrent;
2  using System.Threading.Tasks;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Unsafe
7  {
8      public static unsafe class MemoryBlock
9      {
10         public static void Zero(void* pointer, long capacity)
11         {
12             Parallel.ForEach(Partitioner.Create(0, capacity), range =>

```

```

13         {
14             var from = range.Item1;
15             var offset = (void*)((byte*)pointer + from);
16             var length = (uint)(range.Item2 - from);
17             System.Runtime.CompilerServices.Unsafe.InitBlock(offset, 0, length);
18         });
19     }
20 }
21 }

```

#### ./Platform.Unsafe/Structure.cs

```

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

```

#### ./Platform.Unsafe/StructureExtensions.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Hardware.Cpu;
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 structureSize = System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
15             var bytes = new byte[structureSize];
16             fixed (byte* pointer = bytes)
17             {
18                 obj.CopyTo(pointer, structureSize);
19             }
20             return bytes;
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static void CopyTo<TStruct>(this ref TStruct source, void* destination)
25             where TStruct : struct
26         {
27             var size = System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
28             CopyTo(ref source, destination, size);
29         }
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public static void CopyTo<TStruct>(this ref TStruct source, void* destination, int size)
33             where TStruct : struct
34         {
35             if (CacheLine.Size <= size)
36             {
37                 System.Runtime.CompilerServices.Unsafe.Copy(destination, ref source);
38             }
39             else
40             {

```

```

41         System.Runtime.CompilerServices.Unsafe.CopyBlock(destination,
    ↪     System.Runtime.CompilerServices.Unsafe.AsPointer(ref source), (uint)size);
42     }
43 }
44 }
45 }

```

./Platform.Unsafe.Tests/IntPtrExtensionsTests.cs

```

1  using System;
2  using System.Runtime.InteropServices;
3  using Xunit;
4  using Xunit.Abstractions;
5  using Platform.Diagnostics;
6
7  namespace Platform.Unsafe.Tests
8  {
9      public unsafe class IntPtrExtensionsTests
10     {
11         private const int N = 10000000;
12
13         private readonly ITestOutputHelper _output;
14
15         public IntPtrExtensionsTests(ITestOutputHelper output)
16         {
17             _output = output;
18         }
19
20         [Fact]
21         public void ReadAndWriteOperationsForPointerValuesDelegatesTest()
22         {
23             var pointer = Marshal.AllocHGlobal(sizeof(ulong));
24             ulong result = default;
25             for (var i = 0; i < N; i++)
26             {
27                 result = Delegates(pointer);
28             }
29             Assert.Equal(42UL, result);
30             Marshal.FreeHGlobal(pointer);
31         }
32
33         private static ulong Delegates(IntPtr pointer)
34         {
35             ulong result;
36             IntPtr<ulong>.SetValue(pointer, 42UL);
37             result = IntPtr<ulong>.GetValue(pointer);
38             return result;
39         }
40
41         [Fact]
42         public void ReadAndWriteOperationsForPointerValuesExtensionMethodsTest()
43         {
44             var pointer = Marshal.AllocHGlobal(sizeof(ulong));
45             ulong result = default;
46             for (var i = 0; i < N; i++)
47             {
48                 result = ExtensionMethods(pointer);
49             }
50             Assert.Equal(42UL, result);
51             Marshal.FreeHGlobal(pointer);
52         }
53
54         private static ulong ExtensionMethods(IntPtr pointer)
55         {
56             ulong result;
57             pointer.SetValue(42UL);
58             result = pointer.GetValue<ulong>();
59             return result;
60         }
61
62         [Fact]
63         public void ReadAndWriteOperationsForPointerValuesUnsafeClassMethodsTest()
64         {
65             void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong));
66             ulong result = default;
67             for (var i = 0; i < N; i++)
68             {
69                 result = ReadAndWriteMethods(pointer);
70             }
71             Assert.Equal(42UL, result);
72             Marshal.FreeHGlobal((IntPtr)pointer);

```

```

73     }
74
75     private static ulong ReadAndWriteMethods(void* pointer)
76     {
77         ulong result;
78         System.Runtime.CompilerServices.Unsafe.Write(pointer, 42UL);
79         result = System.Runtime.CompilerServices.Unsafe.Read<ulong>(pointer);
80         return result;
81     }
82
83     [Fact]
84     public void ReadAndWriteOperationsComparisionTest()
85     {
86         var t1 = Performance.Measure(ReadAndWriteOperationsForPointerValuesDelegatesTest);
87         var t2 =
88             ↪ Performance.Measure(ReadAndWriteOperationsForPointerValuesExtensionMethodsTest);
89         var t3 = Performance.Measure(ReadAndWriteOperationsForPointerValuesUnsafeClassMethod
90             ↪ sTest);
91         var message = $"{t1} {t2} {t3}";
92         _output.WriteLine(message);
93     }
94
95     [Fact]
96     public void ElementOffsetOperationsForPointerValuesExtensionMethods()
97     {
98         var pointer = Marshal.AllocHGlobal(sizeof(ulong) * 10);
99         ulong result = default;
100         for (var i = 0; i < N; i++)
101         {
102             result = GetElementExtensionMethods(pointer);
103         }
104         Assert.Equal(5UL * 8UL, result - (ulong)pointer);
105         Marshal.FreeHGlobal(pointer);
106     }
107
108     private static ulong GetElementExtensionMethods(IntPtr pointer)
109     {
110         ulong result;
111         result = (ulong)pointer.GetElement(8, 5);
112         return result;
113     }
114
115     [Fact]
116     public void ElementOffsetOperationsForPointerValuesUnsafeClassMethodsTest()
117     {
118         void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong) * 10);
119         ulong result = default;
120         for (var i = 0; i < N; i++)
121         {
122             result = GetElementMethods(pointer);
123         }
124         Assert.Equal(5UL * 8UL, result - (ulong)pointer);
125         Marshal.FreeHGlobal((IntPtr)pointer);
126     }
127
128     private static ulong GetElementMethods(void* pointer)
129     {
130         ulong result;
131         result = (ulong)System.Runtime.CompilerServices.Unsafe.Add<ulong>(pointer, 5);
132         return result;
133     }
134
135     [Fact]
136     public void GetElementOperationsComparisionTest()
137     {
138         var t1 =
139             ↪ Performance.Measure(ElementOffsetOperationsForPointerValuesExtensionMethods);
140         var t2 = Performance.Measure(ElementOffsetOperationsForPointerValuesUnsafeClassMetho
141             ↪ dsTest);
142         var message = $"{t1} {t2}";
143         _output.WriteLine(message);
144     }
145 }

```

./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 }

```

./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 }

```

./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

- ./Platform.Unsafe.Tests/IntPtrExtensionsTests.cs, 4
- ./Platform.Unsafe.Tests/SizeOfTests.cs, 5
- ./Platform.Unsafe.Tests/StructAndBytesConversionTests.cs, 6
- ./Platform.Unsafe.Tests/ZeroMemoryTests.cs, 6
- ./Platform.Unsafe/ByteArrayExtensions.cs, 1
- ./Platform.Unsafe/IntPtr.cs, 1
- ./Platform.Unsafe/IntPtrExtensions.cs, 2
- ./Platform.Unsafe/MemoryBlock.cs, 2
- ./Platform.Unsafe/Structure.cs, 3
- ./Platform.Unsafe/StructureExtensions.cs, 3