LinksPlatform's Platform.Unsafe Class Library

./Platform.Unsafe/ByteArrayExtensions.cs

```csharp
using Platform.Exceptions;
using Platform.Collections;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Unsafe
{
    public unsafe static class ByteArrayExtensions
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TStruct ToStructure<TStruct>(this byte[] bytes)
            where TStruct : struct
        {
            Ensure.OnDebug.ArgumentNotEmpty(bytes, nameof(bytes));
            var structureSize = System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
            Ensure.OnDebug.ArgumentMeetsCriteria(bytes, array => array.Length == structureSize,
                nameof(bytes), "Bytes array should be the same length as struct size.");
            TStruct structure = default;
            fixed (byte* pointer = bytes)
            {
                System.Runtime.CompilerServices.Unsafe.Copy(ref structure, pointer);
            }
            return structure;
        }
    }
}
```

./Platform.Unsafe/IntPtr.cs

```csharp
using System;
using System.Reflection;
using System.Runtime.InteropServices;
using Platform.Reflection;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Unsafe
{
    /// <remarks>
    /// Please use System.Runtime.CompilerServices.Unsafe instead.
    /// </remarks>
    public static class IntPtr<T>
    {
        public static readonly Func<IntPtr, T> GetValue;
        public static readonly Action<IntPtr, T> SetValue;

        static IntPtr()
        {
            GetValue = CompileGetValueDelegate();
            SetValue = CompileSetValueDelegate();
        }

        static private Func<IntPtr, T> CompileGetValueDelegate()
        {
            return DelegateHelpers.Compile<Func<IntPtr, T>>(emiter =>
            {
                if (NumericType<T>.IsNumeric)
                {
                    emiter.LoadArgument(0);
                    emiter.LoadIndirect<T>();
                    emiter.Return();
                }
                else
                {
                    emiter.LoadArguments(0);
                    emiter.Call(typeof(Marshal).GetGenericMethod(nameof(Marshal.PtrToStructure),
                        Types<T>.Array, Types<IntPtr, Type, bool>.Array));
                    emiter.Return();
                }
            });
        }

        static private Action<IntPtr, T> CompileSetValueDelegate()
        {
            return DelegateHelpers.Compile<Action<IntPtr, T>>(emiter =>
            {
                if (NumericType<T>.IsNumeric)
```

```
48              {
49                  emiter.LoadArguments(0, 1);
50                  emiter.StoreIndirect<T>();
51                  emiter.Return();
52              }
53              else
54              {
55                  emiter.LoadArguments(0, 1);
56                  emiter.LoadConstant(true);
57                  emiter.Call(typeof(Marshal).GetTypeInfo().GetMethod(nameof(Marshal.Structure⌋
                    ↪  ToPtr), Types<object, IntPtr,
                    ↪  bool>.Array));
58                  emiter.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
             ↪  cref="Marshal.SizeOf{T}()"/> and <see cref="Marshal.SizeOf(Type)"/> do.
15          /// </para>
16          /// <para>
17          /// Возвращает размер неуправляемого типа в байтах.
18          /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
             ↪  это делают <see cref="Marshal.SizeOf{T}()"/> и <see cref="Marshal.SizeOf(Type)"/>.
19          /// </para>
20          /// </summary>
21          public static int Size { get; } =
             ↪  System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
22      }
23  }
```

./Platform.Unsafe/StructureExtensions.cs
```
1   using System.Runtime.CompilerServices;
2
3   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5   namespace Platform.Unsafe
6   {
7       public unsafe static class StructureExtensions
8       {
9           [MethodImpl(MethodImplOptions.AggressiveInlining)]
10          public static byte[] ToBytes<TStruct>(this ref TStruct obj)
11              where TStruct : struct
12          {
13              var structureSize = System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
14              var bytes = new byte[structureSize];
15              fixed (byte* pointer = bytes)
16              {
17                  System.Runtime.CompilerServices.Unsafe.Copy(pointer, ref obj);
18              }
19              return bytes;
20          }
21      }
22  }
```

./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          {
```

```csharp
            _output = output;
        }

        [Fact]
        public void ReadAndWriteOperationsForPointerValuesDelegatesTest()
        {
            var pointer = Marshal.AllocHGlobal(sizeof(ulong));
            ulong result = default;
            for (var i = 0; i < N; i++)
            {
                result = Delegates(pointer);
            }
            Assert.Equal(42UL, result);
            Marshal.FreeHGlobal(pointer);
        }

        private ulong Delegates(IntPtr pointer)
        {
            ulong result;
            IntPtr<ulong>.SetValue(pointer, 42UL);
            result = IntPtr<ulong>.GetValue(pointer);
            return result;
        }

        [Fact]
        public void ReadAndWriteOperationsForPointerValuesExtensionMethodsTest()
        {
            var pointer = Marshal.AllocHGlobal(sizeof(ulong));
            ulong result = default;
            for (var i = 0; i < N; i++)
            {
                result = ExtensionMethods(pointer);
            }
            Assert.Equal(42UL, result);
            Marshal.FreeHGlobal(pointer);
        }

        private ulong ExtensionMethods(IntPtr pointer)
        {
            ulong result;
            pointer.SetValue(42UL);
            result = pointer.GetValue<ulong>();
            return result;
        }

        [Fact]
        public void ReadAndWriteOperationsForPointerValuesUnsafeClassMethodsTest()
        {
            void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong));
            ulong result = default;
            for (var i = 0; i < N; i++)
            {
                result = ReadAndWriteMethods(pointer);
            }
            Assert.Equal(42UL, result);
            Marshal.FreeHGlobal((IntPtr)pointer);
        }

        private ulong ReadAndWriteMethods(void* pointer)
        {
            ulong result;
            System.Runtime.CompilerServices.Unsafe.Write(pointer, 42UL);
            result = System.Runtime.CompilerServices.Unsafe.Read<ulong>(pointer);
            return result;
        }

        [Fact]
        public void ReadAndWriteOperationsComparisionTest()
        {
            var t1 = Performance.Measure(ReadAndWriteOperationsForPointerValuesDelegatesTest);
            var t2 =
              ↪  Performance.Measure(ReadAndWriteOperationsForPointerValuesExtensionMethodsTest);
            var t3 = Performance.Measure(ReadAndWriteOperationsForPointerValuesUnsafeClassMethod↵
              ↪  sTest);

            var message = $"{t1} {t2} {t3}";
            _output.WriteLine(message);
        }
```

```csharp
            [Fact]
            public void ElementOffsetOperationsForPointerValuesExtensionMethods()
            {
                var pointer = Marshal.AllocHGlobal(sizeof(ulong) * 10);
                ulong result = default;
                for (var i = 0; i < N; i++)
                {
                    result = GetElementExtensionMethods(pointer);
                }
                Assert.Equal(5UL * 8UL, result - (ulong)pointer);
                Marshal.FreeHGlobal(pointer);
            }

            private ulong GetElementExtensionMethods(IntPtr pointer)
            {
                ulong result;
                result = (ulong)pointer.GetElement(8, 5);
                return result;
            }

            [Fact]
            public void ElementOffsetOperationsForPointerValuesUnsafeClassMethodsTest()
            {
                void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong) * 10);
                ulong result = default;
                for (var i = 0; i < N; i++)
                {
                    result = GetElementMethods(pointer);
                }
                Assert.Equal(5UL * 8UL, result - (ulong)pointer);
                Marshal.FreeHGlobal((IntPtr)pointer);
            }

            private ulong GetElementMethods(void* pointer)
            {
                ulong result;
                result = (ulong)System.Runtime.CompilerServices.Unsafe.Add<ulong>(pointer, 5);
                return result;
            }

            [Fact]
            public void GetElementOperationsComparisionTest()
            {
                var t1 =
                ↪   Performance.Measure(ElementOffsetOperationsForPointerValuesExtensionMethods);
                var t2 = Performance.Measure(ElementOffsetOperationsForPointerValuesUnsafeClassMetho
                ↪   dsTest);

                var message = $"{t1} {t2}";
                _output.WriteLine(message);
            }
        }
    }
```

./Platform.Unsafe.Tests/SizeOfTests.cs

```csharp
using System.Runtime.InteropServices;
using Xunit;

namespace Platform.Unsafe.Tests
{
    public class SizeOfTests
    {
        public struct X<T>
        {
            public readonly T F1;
            public readonly T F2;
        }

        [Fact]
        public void UnsafeClassSizeOfTest()
        {
            var size = System.Runtime.CompilerServices.Unsafe.SizeOf<X<int>>();
            Assert.Equal(8, size);
        }

        [Fact]
        public void MarshalSizeOfTest()
        {
            var size = Marshal.SizeOf(default(X<int>));
```

```
25              Assert.Equal(8, size);
26          }
27
28          [Fact]
29          public 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 class StructAndBytesConversionTests
6       {
7           [Fact]
8           public 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 void BytesToStructTest()
20          {
21              byte[] bytes = new[] { byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue,
                ↪  byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue };
22              ulong result = bytes.ToStructure<ulong>();
23              Assert.Equal(ulong.MaxValue, result);
24          }
25      }
26  }
```

./Platform.Unsafe.Tests/ZeroMemoryTests.cs

```
1   using Xunit;
2
3   namespace Platform.Unsafe.Tests
4   {
5       public unsafe class ZeroMemoryTests
6       {
7           [Fact]
8           public 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