LinksPlatform's Platform.Unsafe Class Library

1.1 ./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;
        }
    }
}
```

1.2 ./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>
    [Obsolete("Please use System.Runtime.CompilerServices.Unsafe instead.")]
    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 =>
            {
```

```
48          if (NumericType<T>.IsNumeric)
49          {
50              emiter.LoadArguments(0, 1);
51              emiter.StoreIndirect<T>();
52              emiter.Return();
53          }
54          else
55          {
56              emiter.LoadArguments(0, 1);
57              emiter.LoadConstant(true);
58              emiter.Call(typeof(Marshal).GetTypeInfo().GetMethod(nameof(Marshal.Structure
      ↪  ToPtr), Types<object, IntPtr,
      ↪  bool>.Array));
59              emiter.Return();
60          }
61      });
62      }
63  }
64 }
```

1.3 ./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 unsafe 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 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         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public static void WriteElementValue<TValue>(this IntPtr pointer, long index, TValue
           ↪  value) => System.Runtime.CompilerServices.Unsafe.Write((byte*)pointer +
           ↪  (System.Runtime.CompilerServices.Unsafe.SizeOf<TValue>() * index), value);
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public static TValue ReadElementValue<TValue>(this IntPtr pointer, long index) =>
           ↪  System.Runtime.CompilerServices.Unsafe.Read<TValue>((byte*)pointer +
           ↪  (System.Runtime.CompilerServices.Unsafe.SizeOf<TValue>() * index));
39     }
40 }
```

1.4 ./Platform.Unsafe/MemoryBlock.cs

```csharp
using System.Collections.Concurrent;
using System.Threading.Tasks;

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

namespace Platform.Unsafe
{
    public static unsafe class MemoryBlock
    {
        public static void Zero(void* pointer, long capacity)
        {
            Parallel.ForEach(Partitioner.Create(0, capacity), range =>
            {
                var from = range.Item1;
                var offset = (void*)((byte*)pointer + from);
                var length = (uint)(range.Item2 - from);
                System.Runtime.CompilerServices.Unsafe.InitBlock(offset, 0, length);
            });
        }
    }
}
```

1.5 ./Platform.Unsafe/Structure.cs

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

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

namespace Platform.Unsafe
{
    public static class Structure<TStruct>
        where TStruct : struct
    {
        /// <summary>
        /// <para>
        /// Returns the size of an unmanaged type in bytes.
        /// This property do this without throwing exceptions for generic types as <see
        ///  cref="Marshal.SizeOf{T}()"/> and <see cref="Marshal.SizeOf(Type)"/> do.
        /// </para>
        /// <para>
        /// Возвращает размер неуправляемого типа в байтах.
        /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
        ///  это делают <see cref="Marshal.SizeOf{T}()"/> и <see cref="Marshal.SizeOf(Type)"/>.
        /// </para>
        /// </summary>
        public static int Size { get; } =
            System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
    }
}
```

1.6 ./Platform.Unsafe/StructureExtensions.cs

```csharp
using System.Runtime.CompilerServices;
using Platform.Hardware.Cpu;

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

namespace Platform.Unsafe
{
    public unsafe static class StructureExtensions
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static byte[] ToBytes<TStruct>(this ref TStruct obj)
            where TStruct : struct
        {
            var structureSize = System.Runtime.CompilerServices.Unsafe.SizeOf<TStruct>();
            var bytes = new byte[structureSize];
            fixed (byte* pointer = bytes)
            {
                obj.CopyTo(pointer, structureSize);
            }
            return bytes;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void CopyTo<TStruct>(this ref TStruct source, void* destination)
            where TStruct : struct
        {
            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  }
```

1.7   ./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              System.Runtime.CompilerServices.Unsafe.Write((void*)pointer, 42UL);
38              //result = IntPtr<ulong>.GetValue(pointer);
39              result = System.Runtime.CompilerServices.Unsafe.Read<ulong>((void*)pointer);
40              return result;
41          }
42
43          [Fact]
44          public void ReadAndWriteOperationsForPointerValuesExtensionMethodsTest()
45          {
46              var pointer = Marshal.AllocHGlobal(sizeof(ulong));
47              ulong result = default;
48              for (var i = 0; i < N; i++)
49              {
50                  result = ExtensionMethods(pointer);
51              }
52              Assert.Equal(42UL, result);
53              Marshal.FreeHGlobal(pointer);
54          }
55
56          private static ulong ExtensionMethods(IntPtr pointer)
57          {
58              ulong result;
59              //pointer.SetValue(42UL);
```

```csharp
                System.Runtime.CompilerServices.Unsafe.Write((void*)pointer, 42UL);
                //result = pointer.GetValue<ulong>();
                result = System.Runtime.CompilerServices.Unsafe.Read<ulong>((void*)pointer);
                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 static 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);
            }

            [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 static ulong GetElementExtensionMethods(IntPtr pointer)
            {
                ulong result;
                //result = (ulong)pointer.GetElement(8, 5);
                result = (ulong)System.Runtime.CompilerServices.Unsafe.Add<ulong>((void*)pointer, 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 static ulong GetElementMethods(void* pointer)
            {
                ulong result;
                result = (ulong)System.Runtime.CompilerServices.Unsafe.Add<ulong>(pointer, 5);
                return result;
            }
```

```
137
138            [Fact]
139            public void GetElementOperationsComparisionTest()
140            {
141                var t1 =
                  ↪  Performance.Measure(ElementOffsetOperationsForPointerValuesExtensionMethods);
142                var t2 = Performance.Measure(ElementOffsetOperationsForPointerValuesUnsafeClassMetho ↵
                  ↪  dsTest);
143                var message = $"{t1} {t2}";
144                _output.WriteLine(message);
145            }
146        }
147  }
```

1.8 ./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  }
```

1.9 ./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,
                  ↪  byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue };
22              ulong result = bytes.ToStructure<ulong>();
23              Assert.Equal(ulong.MaxValue, result);
24          }
25      }
26  }
```

```csharp
using Xunit;

namespace Platform.Unsafe.Tests
{
    public static unsafe class ZeroMemoryTests
    {
        [Fact]
        public static void ZeroMemoryTest()
        {
            var bytes = new byte[1024];
            for (int i = 0; i < bytes.Length; i++)
            {
                bytes[i] = unchecked((byte)i);
            }
            fixed (byte* pointer = bytes)
            {
                MemoryBlock.Zero(pointer, bytes.Length);
            }
            for (int i = 0; i < bytes.Length; i++)
            {
                Assert.Equal(0, bytes[i]);
            }
        }
    }
}
```

Index