

LinksPlatform's Platform.Unsafe Class Library

1.1 ./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             var structureSize = SizeOf<TStruct>();
18             Ensure.OnDebug.ArgumentMeetsCriteria(bytes, array => array.Length == structureSize,
19                 ↪ nameof(bytes), "Bytes array should be the same length as struct size.");
20             TStruct structure = default;
21             fixed (byte* pointer = bytes)
22             {
23                 Copy(ref structure, pointer);
24             }
25             return structure;
26         }
27     }
```

1.2 ./Platform.Unsafe/IntPtr.cs

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

```

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

```

1.3 ./Platform.Unsafe/IntPtrExtensions.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Numbers;
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     /// <remarks>
11     /// Please use System.Runtime.CompilerServices.Unsafe instead.
12     /// </remarks>
13     public unsafe static class IntPtrExtensions
14     {
15         [Obsolete("GetValue method is deprecated, please use
        ↪ System.Runtime.CompilerServices.Unsafe.Read method instead.")]
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static TElement GetValue<TElement>(this IntPtr pointer) =>
18             ↪ IntPtr<TElement>.GetValue(pointer);
19
20         [Obsolete("SetValue method is deprecated, please use
        ↪ System.Runtime.CompilerServices.Unsafe.Write method instead.")]
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public static void SetValue<TElement>(this IntPtr pointer, TElement value) =>
23             ↪ IntPtr<TElement>.SetValue(pointer, value);
24
25         [Obsolete("GetElement method is deprecated, please use
        ↪ System.Runtime.CompilerServices.Unsafe.Add method instead.")]
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public static IntPtr GetElement(this IntPtr pointer, int elementSize, int index) =>
28             ↪ pointer + (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(this IntPtr pointer, long elementSize, long index) =>
33             ↪ new IntPtr((byte*)pointer.ToPointer() + (elementSize * index));
34
35         [Obsolete("GetElement method is deprecated, please use
        ↪ System.Runtime.CompilerServices.Unsafe.Add method instead.")]
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static IntPtr GetElement<TIndex>(this IntPtr pointer, int elementSize, TIndex
        ↪ index) => pointer.GetElement((long)elementSize, (Integer)(Integer<TIndex>)index);
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public static void WriteElementValue<TValue>(this IntPtr pointer, long index, TValue
        ↪ value) => Write((byte*)pointer + (SizeOf<TValue>() * index), value);
41
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public static TValue ReadElementValue<TValue>(this IntPtr pointer, long index) =>
44             ↪ Read<TValue>((byte*)pointer + (SizeOf<TValue>() * index));
45     }
46 }

```

1.4 ./Platform.Unsafe/MemoryBlock.cs

```

1  using System.Collections.Concurrent;
2  using System.Runtime.CompilerServices;
3  using System.Threading.Tasks;
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 unsafe class MemoryBlock
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static void Zero(void* pointer, long capacity) => InitBlock(pointer, 0,
14             ↳ (uint)capacity);
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static void ParallelZero(void* pointer, long capacity) =>
18             ↳ Parallel.ForEach(Partitioner.Create(0, capacity), range => ZeroBlock(pointer,
19             ↳ range.Item1, range.Item2));
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         private static void ZeroBlock(void* pointer, long from, long to)
23         {
24             var offset = (void*)((byte*)pointer + from);
25             var length = (uint)(to - from);
26             InitBlock(offset, 0, length);
27         }
28     }
29 }

```

1.5 ./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
17         ↳ cref="Marshal.SizeOf{T}()" /> and <see cref="Marshal.SizeOf(Type)" /> do.
18         /// </para>
19         /// <para>
20         /// Возвращает размер неуправляемого типа в байтах.
21         /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
22         ↳ это делают <see cref="Marshal.SizeOf{T}()" /> и <see cref="Marshal.SizeOf(Type)" />.
23         /// </para>
24         /// </summary>
25         public static int Size
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get;
29         } = SizeOf<TStruct>();
30     }
31 }

```

1.6 ./Platform.Unsafe/StructureExtensions.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Hardware.Cpu;
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 StructureExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static byte[] ToBytes<TStruct>(this ref TStruct obj)
13             where TStruct : struct
14         {
15             var structureSize = SizeOf<TStruct>();
16             var bytes = new byte[structureSize];
17             fixed (byte* pointer = bytes)
18             {
19                 obj.CopyTo(pointer, structureSize);
20             }
21         }
22     }
23 }

```

```

20     }
21     return bytes;
22 }
23
24 [MethodImpl(MethodImplOptions.AggressiveInlining)]
25 public static void CopyTo<TStruct>(this ref TStruct source, void* destination)
26     where TStruct : struct
27 {
28     var size = SizeOf<TStruct>();
29     CopyTo(ref source, destination, size);
30 }
31
32 [MethodImpl(MethodImplOptions.AggressiveInlining)]
33 public static void CopyTo<TStruct>(this ref TStruct source, void* destination, int size)
34     where TStruct : struct
35 {
36     if (CacheLine.Size >= size)
37     {
38         Copy(destination, ref source);
39     }
40     else
41     {
42         CopyBlock(destination, AsPointer(ref source), (uint)size);
43     }
44 }
45 }
46 }

```

1.7 ./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 }

```

1.8 ./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,
22                                     ↪ byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue };
23             ulong result = bytes.ToStructure<ulong>();
24             Assert.Equal(ulong.MaxValue, result);
25         }
26     }
27 }

```

1.10 ./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         [Fact]
26         public static void ParallelZeroMemoryTest()
27         {
28             var bytes = new byte[1024];
29             for (int i = 0; i < bytes.Length; i++)
30             {
31                 bytes[i] = unchecked((byte)i);
32             }
33         }
34     }
35 }

```

```
33         fixed (byte* pointer = bytes)
34         {
35             MemoryBlock.ParallelZero(pointer, bytes.Length);
36         }
37         for (int i = 0; i < bytes.Length; i++)
38         {
39             Assert.Equal(0, bytes[i]);
40         }
41     }
42 }
43 }
```

Index

- ./Platform.Unsafe.Tests/IntPtrExtensionsTests.cs, 4
- ./Platform.Unsafe.Tests/SizeOfTests.cs, 4
- ./Platform.Unsafe.Tests/StructAndBytesConversionTests.cs, 5
- ./Platform.Unsafe.Tests/ZeroMemoryTests.cs, 5
- ./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