

LinksPlatform's Platform.Unsafe Class Library

1.1 ./csharp/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     /// <summary>
11     /// <para>
12     /// Represents the byte array extensions.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public unsafe static class ByteArrayExtensions
17     {
18         /// <summary>
19         /// <para>
20         /// Returns the structure using the specified bytes.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <typeparam name="TStruct">
25         /// <para>The struct.</para>
26         /// <para></para>
27         /// </typeparam>
28         /// <param name="bytes">
29         /// <para>The bytes.</para>
30         /// <para></para>
31         /// </param>
32         /// <returns>
33         /// <para>The structure.</para>
34         /// <para></para>
35         /// </returns>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static TStruct ToStructure<TStruct>(this byte[] bytes)
38             where TStruct : struct
39         {
40             Ensure.OnDebug.ArgumentNotEmpty(bytes, nameof(bytes));
41             Ensure.OnDebug.ArgumentMeetsCriteria(bytes, HasSameSizeAs<TStruct>, nameof(bytes),
42                 ↪ "Bytes array should be the same length as struct size.");
43             TStruct structure = default;
44             fixed (byte* pointer = bytes)
45             {
46                 Copy(ref structure, pointer);
47             }
48             return structure;
49         }
50         /// <summary>
51         /// <para>
52         /// Determines whether has same size as.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <typeparam name="TStruct">
57         /// <para>The struct.</para>
58         /// <para></para>
59         /// </typeparam>
60         /// <param name="array">
61         /// <para>The array.</para>
62         /// <para></para>
63         /// </param>
64         /// <returns>
65         /// <para>The bool</para>
66         /// <para></para>
67         /// </returns>
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         private static bool HasSameSizeAs<TStruct>(byte[] array) where TStruct : struct =>
70             ↪ array.Length == Structure<TStruct>.Size;
71     }
```

1.2 ./csharp/Platform.Unsafe/IntPtrExtensions.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
```

```

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     /// <summary>
10    /// <para>
11    /// Represents the int ptr extensions.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    public unsafe static class IntPtrExtensions
16    {
17        /// <summary>
18        /// <para>
19        /// Writes the element value using the specified pointer.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <typeparam name="TValue">
24        /// <para>The value.</para>
25        /// <para></para>
26        /// </typeparam>
27        /// <param name="pointer">
28        /// <para>The pointer.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="index">
32        /// <para>The index.</para>
33        /// <para></para>
34        /// </param>
35        /// <param name="value">
36        /// <para>The value.</para>
37        /// <para></para>
38        /// </param>
39        [MethodImpl(MethodImplOptions.AggressiveInlining)]
40        public static void WriteElementValue<TValue>(this IntPtr pointer, long index, TValue
41        ↪ value) => Write((byte*)pointer + (SizeOf<TValue>() * index), value);
42
43        /// <summary>
44        /// <para>
45        /// Reads the element value using the specified pointer.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        /// <typeparam name="TValue">
50        /// <para>The value.</para>
51        /// <para></para>
52        /// </typeparam>
53        /// <param name="pointer">
54        /// <para>The pointer.</para>
55        /// <para></para>
56        /// </param>
57        /// <param name="index">
58        /// <para>The index.</para>
59        /// <para></para>
60        /// </param>
61        /// <returns>
62        /// <para>The value</para>
63        /// <para></para>
64        /// </returns>
65        [MethodImpl(MethodImplOptions.AggressiveInlining)]
66        public static TValue ReadElementValue<TValue>(this IntPtr pointer, long index) =>
67        ↪ Read<TValue>((byte*)pointer + (SizeOf<TValue>() * index));
68    }
69 }

```

1.3 ./csharp/Platform.Unsafe/MemoryBlock.cs

```

1 using System;
2 using System.Collections.Concurrent;
3 using System.Runtime.CompilerServices;
4 using System.Threading.Tasks;
5 using static System.Runtime.CompilerServices.Unsafe;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Unsafe
10 {

```

```

11  /// <summary>
12  /// <para>
13  /// Represents the memory block.
14  /// </para>
15  /// <para></para>
16  /// </summary>
17  public static unsafe class MemoryBlock
18  {
19      /// <summary>
20      /// <para>
21      /// Zeroes the pointer.
22      /// </para>
23      /// <para></para>
24      /// </summary>
25      /// <param name="pointer">
26      /// <para>The pointer.</para>
27      /// <para></para>
28      /// </param>
29      /// <param name="capacity">
30      /// <para>The capacity.</para>
31      /// <para></para>
32      /// </param>
33      [MethodImpl(MethodImplOptions.AggressiveInlining)]
34      public static void Zero(void* pointer, long capacity)
35      {
36          // A way to prevent wasting resources due to Hyper-Threading.
37          var threads = Environment.ProcessorCount / 2;
38          if (threads <= 1)
39          {
40              ZeroBlock(pointer, 0, capacity);
41          }
42          else
43          {
44              // Using 2 threads because two-channel memory architecture is the most available
45              // ↪ type.
46              // CPUs mostly just wait for memory here.
47              threads = 2;
48              Parallel.ForEach(Partitioner.Create(0L, capacity), new ParallelOptions {
49                  ↪ MaxDegreeOfParallelism = threads }, range => ZeroBlock(pointer, range.Item1,
50                  ↪ range.Item2));
51          }
52      }
53
54      /// <summary>
55      /// <para>
56      /// Zeroes the block using the specified pointer.
57      /// </para>
58      /// <para></para>
59      /// </summary>
60      /// <param name="pointer">
61      /// <para>The pointer.</para>
62      /// <para></para>
63      /// </param>
64      /// <param name="from">
65      /// <para>The from.</para>
66      /// <para></para>
67      /// </param>
68      /// <param name="to">
69      /// <para>The to.</para>
70      /// <para></para>
71      /// </param>
72      [MethodImpl(MethodImplOptions.AggressiveInlining)]
73      private static void ZeroBlock(void* pointer, long from, long to)
74      {
75          var offset = (byte*)pointer + from;
76          var length = to - from;
77          var uintMaxValue = uint.MaxValue;
78          while (length > uintMaxValue)
79          {
80              InitBlock(offset, 0, uintMaxValue);
81              length -= uintMaxValue;
82              offset += uintMaxValue;
83          }
84          InitBlock(offset, 0, unchecked((uint)length));
85      }
86  }

```

1.4 ./csharp/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     /// <summary>
11     /// <para>
12     /// Represents the structure.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public static class Structure<TStruct>
17         where TStruct : struct
18     {
19         /// <summary>
20         /// <para>
21         /// Returns the size of an unmanaged type in bytes.
22         /// This property do this without throwing exceptions for generic types as <see
23         /// → cref="Marshal.SizeOf{T}()"> and <see cref="Marshal.SizeOf(Type)"> do.
24         /// </para>
25         /// <para>
26         /// Возвращает размер неуправляемого типа в байтах.
27         /// Этот свойство делает это без выбрасывания исключений для универсальных типов, как
28         /// → это делают <see cref="Marshal.SizeOf{T}()"> и <see cref="Marshal.SizeOf(Type)">.
29         /// </para>
30         /// </summary>
31         public static int Size
32         {
33             [MethodImpl(MethodImplOptions.AggressiveInlining)]
34             get;
35             } = SizeOf<TStruct>();
36     }
```

1.5 ./csharp/Platform.Unsafe/StructureExtensions.cs

```
1 using System.Runtime.CompilerServices;
2 using static System.Runtime.CompilerServices.Unsafe;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Unsafe
7 {
8     /// <summary>
9     /// <para>Represents a set of extension methods for structs.</para>
10    /// <para>Представляет набор методов расширения для структур.</para>
11    /// </summary>
12    public unsafe static class StructureExtensions
13    {
14        /// <summary>
15        /// <para>this process does something</para>
16        /// <para>этот процесс что-то делает</para>
17        /// </summary>
18        [MethodImpl(MethodImplOptions.AggressiveInlining)]
19        public static byte[] ToBytes<TStruct>(this ref TStruct obj)
20            where TStruct : struct
21        {
22            var bytes = new byte[Structure<TStruct>.Size];
23            fixed (byte* pointer = bytes)
24            {
25                Copy(pointer, ref obj);
26            }
27            return bytes;
28        }
29    }
30 }
```

1.6 ./csharp/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     /// <summary>
9     /// <para>
10    /// Represents the int ptr extensions tests.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    public unsafe class IntPtrExtensionsTests
15    {
16        /// <summary>
17        /// <para>
18        /// Tests that read and write operations for pointer values unsafe class methods test.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        [Fact]
23        public void ReadAndWriteOperationsForPointerValuesUnsafeClassMethodsTest()
24        {
25            void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong));
26            Write(pointer, 42UL);
27            Assert.Equal(42UL, Read<ulong>(pointer));
28            Marshal.FreeHGlobal((IntPtr)pointer);
29        }
30
31        /// <summary>
32        /// <para>
33        /// Tests that element offset operations for pointer values test.
34        /// </para>
35        /// <para></para>
36        /// </summary>
37        [Fact]
38        public void ElementOffsetOperationsForPointerValuesTest()
39        {
40            void* pointer = (void*)Marshal.AllocHGlobal(sizeof(ulong) * 10);
41            ulong result = (ulong)Add<ulong>(pointer, 5);
42            Assert.Equal(5UL * 8UL, result - (ulong)pointer);
43            Marshal.FreeHGlobal((IntPtr)pointer);
44        }
45    }
46 }

```

1.7 ./csharp/Platform.Unsafe.Tests/SizeOfTests.cs

```

1  using System.Runtime.InteropServices;
2  using Xunit;
3
4  namespace Platform.Unsafe.Tests
5  {
6      /// <summary>
7      /// <para>
8      /// Represents the size of tests.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public static class SizeOfTests
13     {
14         /// <summary>
15         /// <para>
16         /// The .
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         public struct X<T>
21         {
22             /// <summary>
23             /// <para>
24             /// The .
25             /// </para>
26             /// <para></para>
27             /// </summary>
28             public readonly T F1;
29             /// <summary>
30             /// <para>
31             /// The .
32             /// </para>
33             /// <para></para>
34             /// </summary>
35             public readonly T F2;
36         }
37
38         /// <summary>

```

```

39     /// <para>
40     /// Tests that unsafe class size of test.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     [Fact]
45     public static void UnsafeClassSizeOfTest()
46     {
47         var size = System.Runtime.CompilerServices.Unsafe.SizeOf<X<int>>();
48         Assert.Equal(8, size);
49     }
50
51     /// <summary>
52     /// <para>
53     /// Tests that marshal size of test.
54     /// </para>
55     /// <para></para>
56     /// </summary>
57     [Fact]
58     public static void MarshalSizeOfTest()
59     {
60         var size = Marshal.SizeOf(default(X<int>));
61         Assert.Equal(8, size);
62     }
63
64     /// <summary>
65     /// <para>
66     /// Tests that structure property test.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     [Fact]
71     public static void StructurePropertyTest()
72     {
73         var size = Structure<X<int>>.Size;
74         Assert.Equal(8, size);
75     }
76 }
77 }

```

1.8 ./csharp/Platform.Unsafe.Tests/StructAndBytesConversionTests.cs

```

1  using Xunit;
2
3  namespace Platform.Unsafe.Tests
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the struct and bytes conversion tests.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public static class StructAndBytesConversionTests
12     {
13         /// <summary>
14         /// <para>
15         /// Tests that struct to bytes test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void StructToBytesTest()
21         {
22             ulong source = ulong.MaxValue;
23             var result = source.ToBytes();
24             for (int i = 0; i < result.Length; i++)
25             {
26                 Assert.Equal(byte.MaxValue, result[i]);
27             }
28         }
29
30         /// <summary>
31         /// <para>
32         /// Tests that bytes to struct test.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         [Fact]
37         public static void BytesToStructTest()

```

```

38     {
39         byte[] bytes = new[] { byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue,
                                ↪ byte.MaxValue, byte.MaxValue, byte.MaxValue, byte.MaxValue };
40         ulong result = bytes.ToStructure<ulong>();
41         Assert.Equal(ulong.MaxValue, result);
42     }
43 }
44 }

```

1.9 ./csharp/Platform.Unsafe.Tests/ZeroMemoryTests.cs

```

1  using Xunit;
2
3  namespace Platform.Unsafe.Tests
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the zero memory tests.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public static unsafe class ZeroMemoryTests
12     {
13         /// <summary>
14         /// <para>
15         /// Tests that zero memory test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void ZeroMemoryTest()
21         {
22             var bytes = new byte[1024];
23             for (int i = 0; i < bytes.Length; i++)
24             {
25                 bytes[i] = unchecked((byte)i);
26             }
27             fixed (byte* pointer = bytes)
28             {
29                 MemoryBlock.Zero(pointer, bytes.Length);
30             }
31             for (int i = 0; i < bytes.Length; i++)
32             {
33                 Assert.Equal(0, bytes[i]);
34             }
35         }
36     }
37 }

```

Index

- ./csharp/Platform.Unsafe.Tests/IntPtrExtensionsTests.cs, 4
- ./csharp/Platform.Unsafe.Tests/SizeOfTests.cs, 5
- ./csharp/Platform.Unsafe.Tests/StructAndBytesConversionTests.cs, 6
- ./csharp/Platform.Unsafe.Tests/ZeroMemoryTests.cs, 7
- ./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