./ByteArrayExtensions.cs

```
using System.Runtime.InteropServices;
using Platform.Exceptions;
using Platform.Collections;

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

namespace Platform.Unsafe
{
    public static class ByteArrayExtensions
    {
        public static TTStruct ToStructure<TTStruct>(this byte[] bytes)
            where TTStruct : struct
        {
            Ensure.Always.ArgumentNotEmpty(bytes, nameof(bytes));
            var structureSize = Structure<TTStruct>.Size;
            Ensure.Always.ArgumentMeetsCriteria(array => array.Length == structureSize, bytes,
            ↪  nameof(bytes), "Bytes array should be the same length as struct size.");
            var pointer = Marshal.AllocHGlobal(structureSize);
            Marshal.Copy(bytes, 0, pointer, structureSize);
            var structure = Marshal.PtrToStructure<TTStruct>(pointer);
            Marshal.FreeHGlobal(pointer);
            return structure;
        }
    }
}
```

./IntPtr.cs

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

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

namespace Platform.Unsafe
{
    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 (CachedTypeInfo<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 (CachedTypeInfo<T>.IsNumeric)
                {
                    emiter.LoadArguments(0, 1);
                    emiter.StoreIndirect<T>();
                    emiter.Return();
```

```
                }
                else
                {
                    emiter.LoadArguments(0, 1);
                    emiter.LoadConstant(true);
                    emiter.Call(typeof(Marshal).GetTypeInfo().GetMethod(nameof(Marshal.Structure
                    ↪  ToPtr), Types<object, IntPtr,
                    ↪  bool>.Array));
                    emiter.Return();
                }
            });
        }
    }
}
```

./IntPtrExtensions.cs

```
using System;
using System.Runtime.CompilerServices;
using Platform.Numbers;

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

namespace Platform.Unsafe
{
    public static class IntPtrExtensions
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TElement GetValue<TElement>(this IntPtr pointer) =>
        ↪  IntPtr<TElement>.GetValue(pointer);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void SetValue<TElement>(this IntPtr pointer, TElement value) =>
        ↪  IntPtr<TElement>.SetValue(pointer, value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static IntPtr GetElement(this IntPtr pointer, int elementSize, int index) =>
        ↪  pointer + (elementSize * index);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static unsafe IntPtr GetElement(this IntPtr pointer, long elementSize, long
        ↪  index) => new IntPtr((byte*)pointer.ToPointer() + (elementSize * index));

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static IntPtr GetElement<TIndex>(this IntPtr pointer, int elementSize, TIndex
        ↪  index) => pointer.GetElement((long)elementSize, (Integer)(Integer<TIndex>)index);
    }
}
```

./MemoryBlock.cs

```
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)
        {
            var ulongs = capacity / sizeof(ulong);
            Parallel.ForEach(Partitioner.Create(0, ulongs), range =>
            {
                for (long i = range.Item1; i < range.Item2; i++)
                {
                    *((ulong*)pointer + i) = 0;
                }
            });
            for (var i = ulongs * sizeof(ulong); i < capacity; i++)
            {
                *((byte*)pointer + i) = 0;
            }
        }
    }
}
```

./Structure.cs

```
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 method do this without throwing exceptions for generic types as <see cref="Marshal.SizeOf{T}()"/> and <see cref="Marshal.SizeOf(System.Type)"/> do.
        /// </para>
        /// <para>
        /// Возвращает размер неуправляемого типа в байтах.
        /// Этот метод делает это без выбрасывания исключений для универсальных типов, как это делают <see cref="Marshal.SizeOf{T}()"/> и <see cref="Marshal.SizeOf(System.Type)"/>.
        /// </para>
        /// </summary>
        /// <remarks>
        /// <para>
        /// Implemented based on <see href="https://stackoverflow.com/a/18167584/710069">proposed solution</see> at StackOverflow.
        /// </para>
        /// <para>
        /// Actual differences:
        /// <a href="https://github.com/Microsoft/referencesource/blob/f82e13c3820cd04553c21bf6da01262b95d9bd43/mscorlib/system/runtime/interopservices/marshal.cs#L202">code of</a>
        /// <see cref="Marshal.SizeOf(object)"/>
        /// and
        /// <a href="https://github.com/Microsoft/referencesource/blob/f82e13c3820cd04553c21bf6da01262b95d9bd43/mscorlib/system/runtime/interopservices/marshal.cs#L219-L222">code of</a>
        /// <see cref="Marshal.SizeOf(System.Type)"/>.
        /// </para>
        /// <para>
        /// Note that this behaviour can be changed in future versions of .NET.
        /// </para>
        /// <para>
        /// Реализовано на основе <see href="https://stackoverflow.com/a/18167584/710069">предложенного решения</see> в StackOverflow.
        /// </para>
        /// <para>
        /// Фактические различия:
        /// <a href="https://github.com/Microsoft/referencesource/blob/f82e13c3820cd04553c21bf6da01262b95d9bd43/mscorlib/system/runtime/interopservices/marshal.cs#L202">код</a>
        /// <see cref="Marshal.SizeOf(object)"/>
        /// и
        /// <a href="https://github.com/Microsoft/referencesource/blob/f82e13c3820cd04553c21bf6da01262b95d9bd43/mscorlib/system/runtime/interopservices/marshal.cs#L219-L222">код</a>
        /// <see cref="Marshal.SizeOf(System.Type)"/>.
        /// </para>
        /// <para>
        /// Обратите внимание, что это поведение может быть изменено в будущих версиях .NET.
        /// </para>
        /// </remarks>
        public static int Size { get; } = Marshal.SizeOf(default(TStruct));
    }
}
```

./StructureExtensions.cs

```
using System.Runtime.InteropServices;

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

namespace Platform.Unsafe
{
    public static class StructureExtensions
    {
        public static byte[] ToBytes<TStruct>(this TStruct obj)
            where TStruct : struct
        {
```

```
            var structureSize = Structure<TStruct>.Size;
            var bytes = new byte[structureSize];
            var pointer = Marshal.AllocHGlobal(structureSize);
            Marshal.StructureToPtr(obj, pointer, true);
            Marshal.Copy(pointer, bytes, 0, structureSize);
            Marshal.FreeHGlobal(pointer);
            return bytes;
        }
    }
}
```

Index