1.1 ./Platform.Memory/ArrayMemory.cs

```
using System.Runtime.CompilerServices;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block with access via indexer.</para>
    /// <para>Представляет блок памяти с доступом через индексатор.</para>
    /// </summary>
    /// <typeparam name="TElement"><para>Element type.</para><para>Тип
    ↪  элемента.</para></typeparam>
    public class ArrayMemory<TElement> : IArrayMemory<TElement>
    {
        #region Fields

        private readonly TElement[] _array;

        #endregion

        #region Properties

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪  path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
        public long Size
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _array.Length;
        }

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
        ↪  ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
        public TElement this[long index]
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _array[index];
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set => _array[index] = value;
        }

        #endregion

        #region Constuctors

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}"/>
        ↪  class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪  cref="ArrayMemory{TElement}"/>.</para>
        /// </summary>
        /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public ArrayMemory(long size) => _array = new TElement[size];

        #endregion
    }
}
```

1.2 ./Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs

```
using System;
using System.Runtime.CompilerServices;
using Platform.Disposables;
using Platform.Exceptions;
using Platform.Unsafe;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents adapter to a memory block with access via indexer.</para>
    /// <para>Представляет адаптер к блоку памяти с доступом через индексатор.</para>
    /// </summary>
    /// <typeparam name="TElement"><para>Element type.</para><para>Тип
    ↪  элемента.</para></typeparam>
    public class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
    ↪  IArrayMemory<TElement>, IDirectMemory
        where TElement : struct
    {
```

```
17          #region Fields
18
19          private readonly IDirectMemory _memory;
20
21          #endregion
22
23          #region Properties
24
25          /// <inheritdoc/>
26          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
27          public long Size
28          {
29              [MethodImpl(MethodImplOptions.AggressiveInlining)]
30              get => _memory.Size;
31          }
32
33          /// <inheritdoc/>
34          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
35          public IntPtr Pointer
36          {
37              [MethodImpl(MethodImplOptions.AggressiveInlining)]
38              get => _memory.Pointer;
39          }
40
41          /// <inheritdoc/>
42          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪  ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
43          public TElement this[long index]
44          {
45              [MethodImpl(MethodImplOptions.AggressiveInlining)]
46              get => Pointer.ReadElementValue<TElement>(index);
47              [MethodImpl(MethodImplOptions.AggressiveInlining)]
48              set => Pointer.WriteElementValue(index, value);
49          }
50
51          #endregion
52
53          #region DisposableBase Properties
54
55          /// <inheritdoc/>
56          protected override string ObjectName
57          {
58              [MethodImpl(MethodImplOptions.AggressiveInlining)]
59              get => $"Array as memory block at '{Pointer}' address.";
60          }
61
62          #endregion
63
64          #region Constructors
65
66          /// <summary>
67          /// <para>Initializes a new instance of the <see
            ↪  cref="DirectMemoryAsArrayMemoryAdapter{TElement}"/> class.</para>
68          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="DirectMemoryAsArrayMemoryAdapter{TElement}"/>.</para>
69          /// </summary>
70          /// <param name="memory"><para>An object implementing <see cref="IDirectMemory"/>
            ↪  interface.</para><para>Объект, реализующий интерфейс <see
            ↪  cref="IDirectMemory"/>.</para></param>
71          [MethodImpl(MethodImplOptions.AggressiveInlining)]
72          public DirectMemoryAsArrayMemoryAdapter(IDirectMemory memory)
73          {
74              Ensure.Always.ArgumentNotNull(memory, nameof(memory));
75              Ensure.Always.ArgumentMeetsCriteria(memory, m => (m.Size % Structure<TElement>.Size)
                ↪  == 0, nameof(memory), "Memory is not aligned to element size.");
76              _memory = memory;
77          }
78
79          #endregion
80
81          #region DisposableBase Methods
82
83          /// <inheritdoc/>
84          [MethodImpl(MethodImplOptions.AggressiveInlining)]
85          protected override void Dispose(bool manual, bool wasDisposed)
86          {
87              if (!wasDisposed)
```

```
            {
                _memory.DisposeIfPossible();
            }
        }

        #endregion
    }
}
```

1.3 ./Platform.Memory/FileArrayMemory.cs

```
using System.IO;
using System.Runtime.CompilerServices;
using Platform.Disposables;
using Platform.Unsafe;
using Platform.IO;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block with access via indexer and stored as file on disk.</para>
    /// <para>Представляет блок памяти с доступом через индексатор и хранящийся в виде файла на диске.</para>
    /// </summary>
    /// <typeparam name="TElement"><para>Element type.</para><para>Тип элемента.</para></typeparam>
    public class FileArrayMemory<TElement> : DisposableBase, IArrayMemory<TElement> //-V3073
        where TElement : struct
    {
        #region Fields

        private readonly string _address;
        private readonly FileStream _file;

        #endregion

        #region Properties

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
        public long Size
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _file.Length;
        }

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/member[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
        public TElement this[long index]
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get
            {
                _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                return _file.ReadOrDefault<TElement>();
            }
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set
            {
                _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                _file.Write(value);
            }
        }

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => $"File stored memory block at '{_address}' path.";
        }

        #endregion

        #region Contructors
```

```

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}"/>
        ↪  class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪  cref="FileArrayMemory{TElement}"/>.</para>
        /// </summary>
        /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public FileArrayMemory(string path)
        {
            _address = path;
            _file = File.Open(path, FileMode.OpenOrCreate, FileAccess.ReadWrite);
        }

        #endregion

        #region DisposableBase Methods

        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if(!wasDisposed)
            {
                _file.DisposeIfPossible();
            }
        }

        #endregion
    }
}
```

1.4 ./Platform.Memory/FileMappedResizableDirectMemory.cs

```
using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Runtime.CompilerServices;
using Platform.Disposables;
using Platform.Exceptions;
using Platform.Collections;
using Platform.IO;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block stored as a file on disk.</para>
    /// <para>Представляет блок памяти, хранящийся в виде файла на диске.</para>
    /// </summary>
    public unsafe class FileMappedResizableDirectMemory : ResizableDirectMemoryBase
    {
        #region Fields

        private MemoryMappedFile _file;
        private MemoryMappedViewAccessor _accessor;

        /// <summary>
        /// <para>Gets path to memory mapped file.</para>
        /// <para>Получает путь к отображенному в памяти файлу.</para>
        /// </summary>
        protected readonly string Path;

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => $"File stored memory block at '{Path}' path.";
        }

        #endregion

        #region Constructors

        /// <summary>
        /// <para>Initializes a new instance of the <see
        ↪  cref="FileMappedResizableDirectMemory"/> class.</para>
```

```
        /// <para>Инициализирует новый экземпляр класса <see cref="FileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
        /// <param name="minimumReservedCapacity"><para>Minimum file size in bytes.</para><para>Минимальный размер файла в байтах.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public FileMappedResizableDirectMemory(string path, long minimumReservedCapacity)
        {
            Ensure.Always.ArgumentNotEmptyAndNotWhiteSpace(path, nameof(path));
            if (minimumReservedCapacity < MinimumCapacity)
            {
                minimumReservedCapacity = MinimumCapacity;
            }
            Path = path;
            var size = FileHelpers.GetSize(Path);
            ReservedCapacity = size > minimumReservedCapacity ? ((size / minimumReservedCapacity) + 1) * minimumReservedCapacity : minimumReservedCapacity;
            UsedCapacity = size;
        }

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="FileMappedResizableDirectMemory"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see cref="FileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        /// <param name="address"><para>An path to file.</para><para>Путь к файлу.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public FileMappedResizableDirectMemory(string address) : this(address, MinimumCapacity) { }

        #endregion

        #region Methods

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void MapFile(long capacity)
        {
            if (Pointer != IntPtr.Zero)
            {
                return;
            }
            _file = MemoryMappedFile.CreateFromFile(Path, FileMode.Open, mapName: null, capacity, MemoryMappedFileAccess.ReadWrite);
            _accessor = _file.CreateViewAccessor();
            byte* pointer = null;
            _accessor.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
            Pointer = new IntPtr(pointer);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void UnmapFile()
        {
            if (UnmapFile(Pointer))
            {
                Pointer = IntPtr.Zero;
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private bool UnmapFile(IntPtr pointer)
        {
            if (pointer == IntPtr.Zero)
            {
                return false;
            }
            if (_accessor != null)
            {
                _accessor.SafeMemoryMappedViewHandle.ReleasePointer();
                Disposable.TryDisposeAndResetToDefault(ref _accessor);
            }
            Disposable.TryDisposeAndResetToDefault(ref _file);
            return true;
        }

        #endregion

```

```
117         #region ResizableDirectMemoryBase Methods
118
119         /// <inheritdoc/>
120         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
            ↪ edCapacityChanged(System.Int64,System.Int64)"]/*'/>
121         [MethodImpl(MethodImplOptions.AggressiveInlining)]
122         protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
            ↪ newReservedCapacity)
123         {
124             UnmapFile();
125             FileHelpers.SetSize(Path, newReservedCapacity);
126             MapFile(newReservedCapacity);
127         }
128
129         /// <inheritdoc/>
130         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
            ↪ ointer(System.IntPtr,System.Int64)"]/*'/>
131         [MethodImpl(MethodImplOptions.AggressiveInlining)]
132         protected override void DisposePointer(IntPtr pointer, long usedCapacity)
133         {
134             if (UnmapFile(pointer))
135             {
136                 FileHelpers.SetSize(Path, usedCapacity);
137             }
138         }
139
140         #endregion
141     }
142 }
```

1.5 ./Platform.Memory/HeapResizableDirectMemory.cs

```
1  using System;
2  using System.Runtime.CompilerServices;
3  using System.Runtime.InteropServices;
4  using Platform.Unsafe;
5
6  namespace Platform.Memory
7  {
8      /// <summary>
9      /// <para>Represents a memory block allocated in Heap.</para>
10     /// <para>Представляет блок памяти, выделенный в "куче".</para>
11     /// </summary>
12     public unsafe class HeapResizableDirectMemory : ResizableDirectMemoryBase
13     {
14         #region DisposableBase Properties
15
16         /// <inheritdoc/>
17         protected override string ObjectName
18         {
19             [MethodImpl(MethodImplOptions.AggressiveInlining)]
20             get => $"Heap stored memory block at {Pointer} address.";
21         }
22
23         #endregion
24
25         #region Constructors
26
27         /// <summary>
28         /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
            ↪ class.</para>
29         /// <para>Инициализирует новый экземпляр класса <see
            ↪ cref="HeapResizableDirectMemory"/>.</para>
30         /// </summary>
31         /// <param name="minimumReservedCapacity"><para>Minimum file size in
            ↪ bytes.</para><para>Минимальный размер файла в байтах.</para></param>
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public HeapResizableDirectMemory(long minimumReservedCapacity)
34         {
35             if (minimumReservedCapacity < MinimumCapacity)
36             {
37                 minimumReservedCapacity = MinimumCapacity;
38             }
39             ReservedCapacity = minimumReservedCapacity;
40             UsedCapacity = 0;
41         }
42
43         /// <summary>
```

```
44          /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
            ↪  class.</para>
45          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="HeapResizableDirectMemory"/>.</para>
46          /// </summary>
47          [MethodImpl(MethodImplOptions.AggressiveInlining)]
48          public HeapResizableDirectMemory() : this(MinimumCapacity) { }
49
50          #endregion
51
52          #region ResizableDirectMemoryBase Methods
53
54          /// <inheritdoc/>
55          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
            ↪  ointer(System.IntPtr,System.Int64)"]/*'/>
56          [MethodImpl(MethodImplOptions.AggressiveInlining)]
57          protected override void DisposePointer(IntPtr pointer, long usedCapacity) =>
            ↪  Marshal.FreeHGlobal(pointer);
58
59          /// <inheritdoc/>
60          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
            ↪  edCapacityChanged(System.Int64,System.Int64)"]/*'/>
61          [MethodImpl(MethodImplOptions.AggressiveInlining)]
62          protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
            ↪  newReservedCapacity)
63          {
64              if (Pointer == IntPtr.Zero)
65              {
66                  Pointer = Marshal.AllocHGlobal(new IntPtr(newReservedCapacity));
67                  MemoryBlock.Zero((void*)Pointer, newReservedCapacity);
68              }
69              else
70              {
71                  Pointer = Marshal.ReAllocHGlobal(Pointer, new IntPtr(newReservedCapacity));
72                  var pointer = (byte*)Pointer + oldReservedCapacity;
73                  MemoryBlock.Zero(pointer, newReservedCapacity - oldReservedCapacity);
74              }
75          }
76
77          #endregion
78      }
79  }
```

1.6 ./Platform.Memory/IArrayMemory.cs

```
1   using System.Runtime.CompilerServices;
2
3   namespace Platform.Memory
4   {
5       /// <summary>
6       /// <para>Represents a memory block interface with access via indexer.</para>
7       /// <para>Представляет интерфейс блока памяти с доступом через индексатор.</para>
8       /// </summary>
9       /// <typeparam name="TElement"><para>Element type.</para><para>Тип
            ↪  элемента.</para></typeparam>
10      public interface IArrayMemory<TElement> : IMemory
11      {
12          /// <summary>
13          /// <para>Gets or sets the element at the specified index.</para>
14          /// <para>Возвращает или устанавливает элемент по указанному индексу.</para>
15          /// </summary>
16          /// <param name="index"><para>The index of the element to get or set.</para><para>Индекс
            ↪  элемента, который нужно получить или установить.</para></param>
17          TElement this[long index]
18          {
19              [MethodImpl(MethodImplOptions.AggressiveInlining)]
20              get;
21              [MethodImpl(MethodImplOptions.AggressiveInlining)]
22              set;
23          }
24      }
25  }
```

1.7 ./Platform.Memory/IDirectMemory.cs

```
1   using System;
2   using System.Runtime.CompilerServices;
3
```

```
namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block interface with direct access (via unmanaged pointers).</para>
    /// <para>Представляет интерфейс блока памяти с прямым доступом (через неуправляемые указатели).</para>
    /// </summary>
    public interface IDirectMemory : IMemory, IDisposable
    {
        /// <summary>
        /// <para>Gets the pointer to the beginning of this memory block.</para>
        /// <para>Возвращает указатель на начало блока памяти.</para>
        /// </summary>
        IntPtr Pointer
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }
    }
}
```

1.8 ./Platform.Memory/IMemory.cs

```
using System.Runtime.CompilerServices;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block interface with size in bytes.</para>
    /// <para>Представляет интерфейс блока памяти с размером в байтах.</para>
    /// </summary>
    public interface IMemory
    {
        /// <summary>
        /// <para>Gets the size in bytes of this memory block.</para>
        /// <para>Возвращает размер блока памяти в байтах.</para>
        /// </summary>
        long Size
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }
    }
}
```

1.9 ./Platform.Memory/IResizableDirectMemory.cs

```
using System.Runtime.CompilerServices;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a resizable memory block interface with direct access (via unmanaged pointers).</para>
    /// <para>Представляет интерфейс блока памяти с изменяемым размером и прямым доступом (через неуправляемые указатели).</para>
    /// </summary>
    public interface IResizableDirectMemory : IDirectMemory
    {
        /// <summary>
        /// <para>Gets or sets the reserved capacity in bytes of this memory block.</para>
        /// <para>Возвращает или устаналивает зарезервированный размер блока памяти в байтах.</para>
        /// </summary>
        /// <remarks>
        /// <para>
        /// If less then zero the value is replaced with zero.
        /// Cannot be less than the used capacity of this memory block.
        /// </para>
        /// <para>
        /// Если меньше нуля, значение заменяется на ноль.
        /// Не может быть меньше используемой емкости блока памяти.
        /// </para>
        /// </remarks>
        long ReservedCapacity
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
30              set;
31          }
32
33          /// <summary>
34          /// <para>Gets or sets the used capacity in bytes of this memory block.</para>
35          /// <para>Возвращает или устанавливает используемый размер в блоке памяти (в
            ↪  байтах).</para>
36          /// </summary>
37          /// <remarks>
38          /// <para>
39          /// If less then zero the value is replaced with zero.
40          /// Cannot be greater than the reserved capacity of this memory block.
41          /// </para>
42          /// <para>
43          /// It is recommended to reduce the reserved capacity of the memory block to the used
            ↪  capacity (specified in this property) after the completion of the use of the memory
            ↪  block.
44          /// </para>
45          /// <para>
46          /// Если меньше нуля, значение заменяется на ноль.
47          /// Не может быть больше, чем зарезервированная емкость этого блока памяти.
48          /// </para>
49          /// <para>
50          /// Рекомендуется уменьшать фактический размер блока памяти до используемого размера
            ↪  (указанного в этом свойстве) после завершения использования блока памяти.
51          /// </para>
52          /// </remarks>
53          long UsedCapacity
54          {
55              [MethodImpl(MethodImplOptions.AggressiveInlining)]
56              get;
57              [MethodImpl(MethodImplOptions.AggressiveInlining)]
58              set;
59          }
60      }
61  }
```

1.10 ./Platform.Memory/ResizableDirectMemoryBase.cs

```
1   using System;
2   using System.Threading;
3   using System.Runtime.CompilerServices;
4   using Platform.Exceptions;
5   using Platform.Disposables;
6   using Platform.Ranges;
7
8   namespace Platform.Memory
9   {
10      /// <summary>
11      /// <para>Provides a base implementation for the resizable memory block with direct access
            ↪  (via unmanaged pointers).</para>
12      /// <para>Предоставляет базовую реализацию для блока памяти с изменяемым размером и прямым
            ↪  доступом (через неуправляемые указатели).</para>
13      /// </summary>
14      public abstract class ResizableDirectMemoryBase : DisposableBase, IResizableDirectMemory
15      {
16          #region Constants
17
18          /// <summary>
19          /// <para>Gets minimum capacity in bytes.</para>
20          /// <para>Возвращает минимальную емкость в байтах.</para>
21          /// </summary>
22          public static readonly long MinimumCapacity = 4096;
23
24          #endregion
25
26          #region Fields
27
28          private IntPtr _pointer;
29          private long _reservedCapacity;
30          private long _usedCapacity;
31
32          #endregion
33
34          #region Properties
35
36          /// <inheritdoc/>
37          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
38          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
```

```
39          public long Size
40          {
41              [MethodImpl(MethodImplOptions.AggressiveInlining)]
42              get
43              {
44                  Ensure.Always.NotDisposed(this);
45                  return UsedCapacity;
46              }
47          }
48
49          /// <inheritdoc/>
50          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
51          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
52          public IntPtr Pointer
53          {
54              [MethodImpl(MethodImplOptions.AggressiveInlining)]
55              get
56              {
57                  Ensure.Always.NotDisposed(this);
58                  return _pointer;
59              }
60              [MethodImpl(MethodImplOptions.AggressiveInlining)]
61              protected set
62              {
63                  Ensure.Always.NotDisposed(this);
64                  _pointer = value;
65              }
66          }
67
68          /// <inheritdoc/>
69          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪  ber[@name="P:Platform.Memory.IResizableDirectMemory.ReservedCapacity"]/*'/>
70          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
71          /// <exception cref="ArgumentOutOfRangeException"><para>Attempted to set the reserved
            ↪  capacity to a value that is less than the used capacity.</para><para>Была выполнена
            ↪  попытка установить зарезервированную емкость на значение, которое меньше
            ↪  используемой емкости.</para></exception>
72          public long ReservedCapacity
73          {
74              [MethodImpl(MethodImplOptions.AggressiveInlining)]
75              get
76              {
77                  Ensure.Always.NotDisposed(this);
78                  return _reservedCapacity;
79              }
80              [MethodImpl(MethodImplOptions.AggressiveInlining)]
81              set
82              {
83                  Ensure.Always.NotDisposed(this);
84                  if (value != _reservedCapacity)
85                  {
86                      Ensure.Always.ArgumentInRange(value, new Range<long>(_usedCapacity,
                        ↪  long.MaxValue));
87                      OnReservedCapacityChanged(_reservedCapacity, value);
88                      _reservedCapacity = value;
89                  }
90              }
91          }
92
93          /// <inheritdoc/>
94          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪  ber[@name="P:Platform.Memory.IResizableDirectMemory.UsedCapacity"]/*'/>
95          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
96          /// <exception cref="ArgumentOutOfRangeException"><para>Attempted to set the used
            ↪  capacity to a value that is greater than the reserved capacity or less than
            ↪  zero.</para><para>Была выполнена попытка установить используемую емкость на
            ↪  значение, которое больше, чем зарезервированная емкость или меньше
            ↪  нуля.</para></exception>
97          public long UsedCapacity
98          {
99              [MethodImpl(MethodImplOptions.AggressiveInlining)]
100             get
101             {
102                 Ensure.Always.NotDisposed(this);
```

```
103                 return _usedCapacity;
104             }
105             [MethodImpl(MethodImplOptions.AggressiveInlining)]
106             set
107             {
108                 Ensure.Always.NotDisposed(this);
109                 if (value != _usedCapacity)
110                 {
111                     Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
112                     _usedCapacity = value;
113                 }
114             }
115         }
116 
117         #endregion
118 
119         #region DisposableBase Properties
120 
121         /// <inheritdoc/>
122         protected override bool AllowMultipleDisposeCalls
123         {
124             [MethodImpl(MethodImplOptions.AggressiveInlining)]
125             get => true;
126         }
127 
128         #endregion
129 
130         #region Methods
131 
132         /// <summary>
133         /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
            ↪ property.</para>
134         /// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
135         /// </summary>
136         /// <param name="oldReservedCapacity"><para>The old reserved capacity of the memory
            ↪ block in bytes.</para><para>Старая зарезервированная емкость блока памяти в
            ↪ байтах.</para></param>
137         /// <param name="newReservedCapacity"><para>The new reserved capacity of the memory
            ↪ block in bytes.</para><para>Новая зарезервированная емкость блока памяти в
            ↪ байтах.</para></param>
138         [MethodImpl(MethodImplOptions.AggressiveInlining)]
139         protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
            ↪ newReservedCapacity);
140 
141         /// <summary>
142         /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
143         /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
144         /// </summary>
145         /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
            ↪ блок памяти.</para></param>
146         /// <param name="usedCapacity"><para>The used capacity of the memory block in
            ↪ bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
147         [MethodImpl(MethodImplOptions.AggressiveInlining)]
148         protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
149 
150         #endregion
151 
152         #region DisposableBase Methods
153 
154         /// <inheritdoc/>
155         [MethodImpl(MethodImplOptions.AggressiveInlining)]
156         protected override void Dispose(bool manual, bool wasDisposed)
157         {
158             if (!wasDisposed)
159             {
160                 var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
161                 if (pointer != IntPtr.Zero)
162                 {
163                     DisposePointer(pointer, _usedCapacity);
164                 }
165             }
166         }
167 
168         #endregion
169     }
170 }
```

1.11 ./Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs

```
using System.IO;
using System.Runtime.CompilerServices;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block stored as a temporary file on disk.</para>
    /// <para>Представляет блок памяти, хранящийся в виде временного файла на диске.</para>
    /// </summary>
    public class TemporaryFileMappedResizableDirectMemory : FileMappedResizableDirectMemory
    {
        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => $"Temporary file stored memory block at '{Path}' path.";
        }

        #endregion

        #region Constructors

        /// <summary>
        /// <para>Initializes a new instance of the <see
        ↪  cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪  cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        /// <param name="minimumReservedCapacity"><para>Minimum file size in
        ↪  bytes.</para><para>Минимальный размер файла в байтах.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TemporaryFileMappedResizableDirectMemory(long minimumReservedCapacity) :
        ↪  base(System.IO.Path.GetTempFileName(), minimumReservedCapacity) { }

        /// <summary>
        /// <para>Initializes a new instance of the <see
        ↪  cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪  cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TemporaryFileMappedResizableDirectMemory() : this(MinimumCapacity) { }

        #endregion

        #region DisposableBase Methods

        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            base.Dispose(manual, wasDisposed);
            if (!wasDisposed)
            {
                File.Delete(Path);
            }
        }

        #endregion
    }
}
```

1.12 ./Platform.Memory.Tests/HeapResizableDirectMemoryTests.cs

```
using Xunit;

namespace Platform.Memory.Tests
{
    public unsafe class HeapResizableDirectMemoryTests
    {
        [Fact]
        public void CorrectMemoryReallocationTest()
        {
            using var heapMemory = new HeapResizableDirectMemory();
            var value1 = GetLastByte(heapMemory);
            heapMemory.ReservedCapacity *= 2;
            var value2 = GetLastByte(heapMemory);
```

```
            Assert.Equal(value1, value2);
            Assert.Equal(0, value1);
        }

        private static byte GetLastByte(HeapResizableDirectMemory heapMemory)
        {
            var pointer1 = (void*)heapMemory.Pointer;
            return *((byte*)pointer1 + heapMemory.ReservedCapacity - 1);
        }
    }
}
```

Index