./Platform.Memory/ArrayMemory.cs

```
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 => _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]
        {
            get => _array[index];
            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>
        public ArrayMemory(long size) => _array = new TElement[size];

        #endregion
    }
}
```

./Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs

```
using System;
using Platform.Disposables;
using Platform.Exceptions;

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 unsafe class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
    ↪  IArrayMemory<TElement>, IDirectMemory
        where TElement : struct
    {
        #region Fields

        private readonly IDirectMemory _memory;

        #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 => _memory.Size;
```

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

./Platform.Memory/FileArrayMemory.cs

```
1   using System.IO;
2   using Platform.Disposables;
3   using Platform.Unsafe;
4   using Platform.IO;
5
6   namespace Platform.Memory
7   {
8       /// <summary>
9       /// <para>Represents a memory block with access via indexer and stored as file on
            ↪  disk.</para>
10      /// <para>Представляет блок памяти с доступом через индексатор и хранящийся в виде файла на
            ↪  диске.</para>
11      /// </summary>
12      /// <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 => _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]
        {
            get
            {
                _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                return _file.ReadOrDefault<TElement>();
            }
            set
            {
                _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                _file.Write(value);
            }
        }

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName => $"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>
        public FileArrayMemory(string path)
        {
            _address = path;
            _file = File.Open(path, FileMode.OpenOrCreate, FileAccess.ReadWrite);
        }

        #endregion

        #region DisposableBase Methods

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

        #endregion
    }
}
```

./Platform.Memory/FileMappedResizableDirectMemory.cs

```
using System;
using System.IO;
using System.IO.MemoryMappedFiles;
```

```
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 => $"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>
        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>
        public FileMappedResizableDirectMemory(string address) : this(address, MinimumCapacity)
        ↪  { }

        #endregion

        #region Methods

        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);
        }

        private void UnmapFile()
        {
            if (UnmapFile(Pointer))
            {
                Pointer = IntPtr.Zero;
            }
        }

        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

        #region ResizableDirectMemoryBase Methods

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪  path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
        ↪  edCapacityChanged(System.Int64,System.Int64)"]/*'/>
        protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
        ↪  newReservedCapacity)
        {
            UnmapFile();
            FileHelpers.SetSize(Path, newReservedCapacity);
            MapFile(newReservedCapacity);
        }

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪  path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
        ↪  ointer(System.IntPtr,System.Int64)"]/*'/>
        protected override void DisposePointer(IntPtr pointer, long usedCapacity)
        {
            if (UnmapFile(pointer))
            {
                FileHelpers.SetSize(Path, usedCapacity);
            }
        }

        #endregion
    }
}
```

./Platform.Memory/HeapResizableDirectMemory.cs

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

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block allocated in Heap.</para>
    /// <para>Представляет блок памяти, выделенный в "куче".</para>
    /// </summary>
    public unsafe class HeapResizableDirectMemory : ResizableDirectMemoryBase
    {
        #region DisposableBase Properties

        /// <inheritdoc/>
```

```
15          protected override string ObjectName => $"Heap stored memory block at {Pointer}
            ↪  address.";
16
17          #endregion
18
19          #region Constructors
20
21          /// <summary>
22          /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
            ↪  class.</para>
23          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="HeapResizableDirectMemory"/>.</para>
24          /// </summary>
25          /// <param name="minimumReservedCapacity"><para>Minimum file size in
            ↪  bytes.</para><para>Минимальный размер файла в байтах.</para></param>
26          public HeapResizableDirectMemory(long minimumReservedCapacity)
27          {
28              if (minimumReservedCapacity < MinimumCapacity)
29              {
30                  minimumReservedCapacity = MinimumCapacity;
31              }
32              ReservedCapacity = minimumReservedCapacity;
33              UsedCapacity = 0;
34          }
35
36          /// <summary>
37          /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
            ↪  class.</para>
38          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="HeapResizableDirectMemory"/>.</para>
39          /// </summary>
40          public HeapResizableDirectMemory() : this(MinimumCapacity) { }
41
42          #endregion
43
44          #region ResizableDirectMemoryBase Methods
45
46          /// <inheritdoc/>
47          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
            ↪  ointer(System.IntPtr,System.Int64)"]/*'/>
48          protected override void DisposePointer(IntPtr pointer, long usedCapacity) =>
            ↪  Marshal.FreeHGlobal(pointer);
49
50          /// <inheritdoc/>
51          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
            ↪  edCapacityChanged(System.Int64,System.Int64)"]/*'/>
52          protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
            ↪  newReservedCapacity)
53          {
54              if (Pointer == IntPtr.Zero)
55              {
56                  Pointer = Marshal.AllocHGlobal(new IntPtr(newReservedCapacity));
57              }
58              else
59              {
60                  Pointer = Marshal.ReAllocHGlobal(Pointer, new IntPtr(newReservedCapacity));
61              }
62          }
63
64          #endregion
65      }
66  }
```

./Platform.Memory/IArrayMemory.cs

```
1   namespace Platform.Memory
2   {
3       /// <summary>
4       /// <para>Represents a memory block interface with access via indexer.</para>
5       /// <para>Представляет интерфейс блока памяти с доступом через индексатор.</para>
6       /// </summary>
7       /// <typeparam name="TElement"><para>Element type.</para><para>Тип
        ↪  элемента.</para></typeparam>
8       public interface IArrayMemory<TElement> : IMemory
9       {
10          /// <summary>
11          /// <para>Gets or sets the element at the specified index.</para>
```

```
        /// <para>Возвращает или устанавливает элемент по указанному индексу.</para>
        /// </summary>
        /// <param name="index"><para>The index of the element to get or set.</para><para>Индекс
        ↪  элемента, который нужно получить или установить.</para></param>
        TElement this[long index] { get; set; }
    }
}
```

./Platform.Memory/IDirectMemory.cs

```
using System;

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 { get; }
    }
}
```

./Platform.Memory/IMemory.cs

```
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 { get; }
    }
}
```

./Platform.Memory/IResizableDirectMemory.cs

```
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 { get; set; }

        /// <summary>
        /// <para>Gets or sets the used capacity in bytes of this memory block.</para>
        /// <para>Возвращает или устанавливает используемый размер в блоке памяти (в
        ↪  байтах).</para>
```

```
        /// </summary>
        /// <remarks>
        /// <para>
        /// If less then zero the value is replaced with zero.
        /// Cannot be greater than the reserved capacity of this memory block.
        /// </para>
        /// <para>
        /// 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.
        /// </para>
        /// <para>
        /// Если меньше нуля, значение заменяется на ноль.
        /// Не может быть больше, чем зарезервированная емкость этого блока памяти.
        /// </para>
        /// <para>
        /// Рекомендуется уменьшать фактический размер блока памяти до используемого размера (указанного в этом свойстве) после завершения использования блока памяти.
        /// </para>
        /// </remarks>
        long UsedCapacity { get; set; }
    }
}
```

./Platform.Memory/ResizableDirectMemoryBase.cs

```
using System;
using System.Threading;
using Platform.Exceptions;
using Platform.Disposables;
using Platform.Ranges;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Provides a base implementation for the resizable memory block with direct access (via unmanaged pointers).</para>
    /// <para>Предоставляет базовую реализацию для блока памяти с изменяемым размером и прямым доступом (через неуправляемые указатели).</para>
    /// </summary>
    public abstract class ResizableDirectMemoryBase : DisposableBase, IResizableDirectMemory
    {
        #region Constants

        /// <summary>
        /// <para>Gets minimum capacity in bytes.</para>
        /// <para>Возвращает минимальную емкость в байтах.</para>
        /// </summary>
        public static readonly long MinimumCapacity = 4096;

        #endregion

        #region Fields

        private IntPtr _pointer;
        private long _reservedCapacity;
        private long _usedCapacity;

        #endregion

        #region Properties

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
        /// <exception cref="ObjectDisposedException"><para>The memory block is disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
        public long Size
        {
            get
            {
                Ensure.Always.NotDisposed(this);
                return UsedCapacity;
            }
        }

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
        /// <exception cref="ObjectDisposedException"><para>The memory block is disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
```

```
50          public IntPtr Pointer
51          {
52              get
53              {
54                  Ensure.Always.NotDisposed(this);
55                  return _pointer;
56              }
57              protected set
58              {
59                  Ensure.Always.NotDisposed(this);
60                  _pointer = value;
61              }
62          }
63
64          /// <inheritdoc/>
65          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪ ber[@name="P:Platform.Memory.IResizableDirectMemory.ReservedCapacity"]/*'/>
66          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪ disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
67          /// <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>
68          public long ReservedCapacity
69          {
70              get
71              {
72                  Ensure.Always.NotDisposed(this);
73                  return _reservedCapacity;
74              }
75              set
76              {
77                  Ensure.Always.NotDisposed(this);
78                  if (value != _reservedCapacity)
79                  {
80                      Ensure.Always.ArgumentInRange(value, new Range<long>(_usedCapacity,
                        ↪ long.MaxValue));
81                      OnReservedCapacityChanged(_reservedCapacity, value);
82                      _reservedCapacity = value;
83                  }
84              }
85          }
86
87          /// <inheritdoc/>
88          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪ ber[@name="P:Platform.Memory.IResizableDirectMemory.UsedCapacity"]/*'/>
89          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪ disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
90          /// <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>
91          public long UsedCapacity
92          {
93              get
94              {
95                  Ensure.Always.NotDisposed(this);
96                  return _usedCapacity;
97              }
98              set
99              {
100                 Ensure.Always.NotDisposed(this);
101                 if (value != _usedCapacity)
102                 {
103                     Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
104                     _usedCapacity = value;
105                 }
106             }
107         }
108
109         #endregion
110
111         #region DisposableBase Properties
112
113         /// <inheritdoc/>
114         protected override bool AllowMultipleDisposeCalls => true;
115
116         #endregion
```

```
117
118         #region Methods
119
120         /// <summary>
121         /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
            ↪  property.</para>
122         /// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
123         /// </summary>
124         /// <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>
125         /// <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>
126         protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
            ↪  newReservedCapacity);
127
128         /// <summary>
129         /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
130         /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
131         /// </summary>
132         /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
            ↪  блок памяти.</para></param>
133         /// <param name="usedCapacity"><para>The used capacity of the memory block in
            ↪  bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
134         protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
135
136         #endregion
137
138         #region DisposableBase Methods
139
140         /// <inheritdoc/>
141         protected override void Dispose(bool manual, bool wasDisposed)
142         {
143             if (!wasDisposed)
144             {
145                 var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
146                 if (pointer != IntPtr.Zero)
147                 {
148                     DisposePointer(pointer, _usedCapacity);
149                 }
150             }
151         }
152
153         #endregion
154     }
155 }
```

./Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs

```
1  using System.IO;
2
3  namespace Platform.Memory
4  {
5      /// <summary>
6      /// <para>Represents a memory block stored as a temporary file on disk.</para>
7      /// <para>Представляет блок памяти, хранящийся в виде временного файла на диске.</para>
8      /// </summary>
9      public class TemporaryFileMappedResizableDirectMemory : FileMappedResizableDirectMemory
10     {
11         #region DisposableBase Properties
12
13         /// <inheritdoc/>
14         protected override string ObjectName => $"Temporary file stored memory block at
            ↪  '{Path}' path.";
15
16         #endregion
17
18         #region Constructors
19
20         /// <summary>
21         /// <para>Initializes a new instance of the <see
            ↪  cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
22         /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
23         /// </summary>
24         /// <param name="minimumReservedCapacity"><para>Minimum file size in
            ↪  bytes.</para><para>Минимальный размер файла в байтах.</para></param>
25         public TemporaryFileMappedResizableDirectMemory(long minimumReservedCapacity) :
            ↪  base(System.IO.Path.GetTempFileName(), minimumReservedCapacity) { }
```

```
26
27          /// <summary>
28          /// <para>Initializes a new instance of the <see
            ↪  cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
29          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
30          /// </summary>
31          public TemporaryFileMappedResizableDirectMemory() : this(MinimumCapacity) { }
32
33          #endregion
34
35          #region DisposableBase Methods
36
37          /// <inheritdoc/>
38          protected override void Dispose(bool manual, bool wasDisposed)
39          {
40              base.Dispose(manual, wasDisposed);
41              if (!wasDisposed)
42              {
43                  File.Delete(Path);
44              }
45          }
46
47          #endregion
48      }
49  }
```

./Platform.Memory.Tests/HeapResizableDirectMemoryTests.cs

```
1   using Xunit;
2
3   namespace Platform.Memory.Tests
4   {
5       public unsafe class HeapResizableDirectMemoryTests
6       {
7           [Fact]
8           public void CorrectMemoryReallocationTest()
9           {
10              using (var heapMemory = new HeapResizableDirectMemory())
11              {
12                  void* pointer1 = (void*)heapMemory.Pointer;
13                  var value1 = System.Runtime.CompilerServices.Unsafe.Read<byte>((byte*)pointer1 +
                    ↪  heapMemory.ReservedCapacity - 1);
14
15                  heapMemory.ReservedCapacity *= 2;
16
17                  void* pointer2 = (void*)heapMemory.Pointer;
18                  var value2 = System.Runtime.CompilerServices.Unsafe.Read<byte>((byte*)pointer2 +
                    ↪  heapMemory.ReservedCapacity - 1);
19
20                  Assert.Equal(value1, value2);
21                  Assert.Equal(0, value1);
22              }
23          }
24      }
25  }
```

Index