1.1 ./csharp/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 ./csharp/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
    {
        #region Fields
```

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

```
            }
        }

        #endregion
    }
}
```

1.3 ./csharp/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 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 '{_file.Name}' path.";
        }

        #endregion

        #region Contructors

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}"/> class.</para>
```

```
67          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="FileArrayMemory{TElement}"/>.</para>
68          /// </summary>
69          /// <param name="file"><para>File stream.</para><para>Файловый поток.</para></param>
70          [MethodImpl(MethodImplOptions.AggressiveInlining)]
71          public FileArrayMemory(FileStream file) => _file = file;
72
73          /// <summary>
74          /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}"/>
            ↪  class.</para>
75          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="FileArrayMemory{TElement}"/>.</para>
76          /// </summary>
77          /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
78          [MethodImpl(MethodImplOptions.AggressiveInlining)]
79          public FileArrayMemory(string path) : this(File.Open(path, FileMode.OpenOrCreate)) { }
80
81          #endregion
82
83          #region DisposableBase Methods
84
85          /// <inheritdoc/>
86          [MethodImpl(MethodImplOptions.AggressiveInlining)]
87          protected override void Dispose(bool manual, bool wasDisposed)
88          {
89              if (!wasDisposed)
90              {
91                  _file.DisposeIfPossible();
92              }
93          }
94
95          #endregion
96      }
97  }
```

1.4 ./csharp/Platform.Memory/FileMappedResizableDirectMemory.cs

```
1   using System;
2   using System.IO;
3   using System.IO.MemoryMappedFiles;
4   using System.Runtime.CompilerServices;
5   using Platform.Disposables;
6   using Platform.Exceptions;
7   using Platform.Collections;
8   using Platform.IO;
9
10  namespace Platform.Memory
11  {
12      /// <summary>
13      /// <para>Represents a memory block stored as a file on disk.</para>
14      /// <para>Представляет блок памяти, хранящийся в виде файла на диске.</para>
15      /// </summary>
16      public unsafe class FileMappedResizableDirectMemory : ResizableDirectMemoryBase
17      {
18          #region Fields
19          private MemoryMappedFile _file;
20          private MemoryMappedViewAccessor _accessor;
21
22          /// <summary>
23          /// <para>Gets path to memory mapped file.</para>
24          /// <para>Получает путь к отображенному в памяти файлу.</para>
25          /// </summary>
26          protected readonly string Path;
27
28          #endregion
29
30          #region DisposableBase Properties
31
32          /// <inheritdoc/>
33          protected override string ObjectName
34          {
35              [MethodImpl(MethodImplOptions.AggressiveInlining)]
36              get => $"File stored memory block at '{Path}' path.";
37          }
38
39          #endregion
40
41          #region Constructors
42
43          /// <summary>
44          /// <para>Initializes a new instance of the <see
            ↪  cref="FileMappedResizableDirectMemory"/> class.</para>
```

```
45          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="FileMappedResizableDirectMemory"/>.</para>
46          /// </summary>
47          /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
48          /// <param name="minimumReservedCapacity"><para>Minimum file size in
            ↪  bytes.</para><para>Минимальный размер файла в байтах.</para></param>
49          [MethodImpl(MethodImplOptions.AggressiveInlining)]
50          public FileMappedResizableDirectMemory(string path, long minimumReservedCapacity)
51          {
52              Ensure.Always.ArgumentNotEmptyAndNotWhiteSpace(path, nameof(path));
53              if (minimumReservedCapacity < MinimumCapacity)
54              {
55                  minimumReservedCapacity = MinimumCapacity;
56              }
57              Path = path;
58              var size = FileHelpers.GetSize(path);
59              ReservedCapacity = size > minimumReservedCapacity ? ((size /
                ↪  minimumReservedCapacity) + 1) * minimumReservedCapacity :
                ↪  minimumReservedCapacity;
60              UsedCapacity = size;
61          }
62
63          /// <summary>
64          /// <para>Initializes a new instance of the <see
            ↪  cref="FileMappedResizableDirectMemory"/> class.</para>
65          /// <para>Инициализирует новый экземпляр класса <see
            ↪  cref="FileMappedResizableDirectMemory"/>.</para>
66          /// </summary>
67          /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
68          [MethodImpl(MethodImplOptions.AggressiveInlining)]
69          public FileMappedResizableDirectMemory(string path) : this(path, MinimumCapacity) { }
70
71          #endregion
72
73          #region Methods
74          [MethodImpl(MethodImplOptions.AggressiveInlining)]
75          private void MapFile(long capacity)
76          {
77              if (Pointer != IntPtr.Zero)
78              {
79                  return;
80              }
81              _file = MemoryMappedFile.CreateFromFile(Path, FileMode.OpenOrCreate, mapName: null,
                ↪  capacity, MemoryMappedFileAccess.ReadWrite);
82              _accessor = _file.CreateViewAccessor();
83              byte* pointer = null;
84              _accessor.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
85              Pointer = new IntPtr(pointer);
86          }
87          [MethodImpl(MethodImplOptions.AggressiveInlining)]
88          private void UnmapFile()
89          {
90              if (UnmapFile(Pointer))
91              {
92                  Pointer = IntPtr.Zero;
93              }
94          }
95          [MethodImpl(MethodImplOptions.AggressiveInlining)]
96          private bool UnmapFile(IntPtr pointer)
97          {
98              if (pointer == IntPtr.Zero)
99              {
100                 return false;
101             }
102             if (_accessor != null)
103             {
104                 _accessor.SafeMemoryMappedViewHandle.ReleasePointer();
105                 Disposable.TryDisposeAndResetToDefault(ref _accessor);
106             }
107             Disposable.TryDisposeAndResetToDefault(ref _file);
108             return true;
109         }
110
111         #endregion
112
113         #region ResizableDirectMemoryBase Methods
114
115         /// <inheritdoc/>
```

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

1.5 ./csharp/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 ./csharp/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 ./csharp/Platform.Memory/IDirectMemory.cs

```
1   using System;
2   using System.Runtime.CompilerServices;
3
4   namespace Platform.Memory
5   {
```

```
    /// <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 ./csharp/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 ./csharp/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)]
            set;
        }
```

```
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 ./csharp/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 = Environment.SystemPageSize;
23
24          #endregion
25
26          #region Fields
27          private IntPtr _pointer;
28          private long _reservedCapacity;
29          private long _usedCapacity;
30
31          #endregion
32
33          #region Properties
34
35          /// <inheritdoc/>
36          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
37          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
38          public long Size
39          {
40              [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
41              get
42              {
43                  Ensure.Always.NotDisposed(this);
44                  return UsedCapacity;
45              }
46          }
47
48          /// <inheritdoc/>
49          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
            ↪  path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
50          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
51          public IntPtr Pointer
52          {
53              [MethodImpl(MethodImplOptions.AggressiveInlining)]
54              get
55              {
56                  Ensure.Always.NotDisposed(this);
57                  return _pointer;
58              }
59              [MethodImpl(MethodImplOptions.AggressiveInlining)]
60              protected set
61              {
62                  Ensure.Always.NotDisposed(this);
63                  _pointer = value;
64              }
65          }
66
67          /// <inheritdoc/>
68          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪  ber[@name="P:Platform.Memory.IResizableDirectMemory.ReservedCapacity"]/*'/>
69          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
70          /// <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>
71          public long ReservedCapacity
72          {
73              [MethodImpl(MethodImplOptions.AggressiveInlining)]
74              get
75              {
76                  Ensure.Always.NotDisposed(this);
77                  return _reservedCapacity;
78              }
79              [MethodImpl(MethodImplOptions.AggressiveInlining)]
80              set
81              {
82                  Ensure.Always.NotDisposed(this);
83                  if (value != _reservedCapacity)
84                  {
85                      Ensure.Always.ArgumentInRange(value, new Range<long>(_usedCapacity,
                        ↪  long.MaxValue));
86                      OnReservedCapacityChanged(_reservedCapacity, value);
87                      _reservedCapacity = value;
88                  }
89              }
90          }
91
92          /// <inheritdoc/>
93          /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
            ↪  ber[@name="P:Platform.Memory.IResizableDirectMemory.UsedCapacity"]/*'/>
94          /// <exception cref="ObjectDisposedException"><para>The memory block is
            ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
95          /// <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>
96          public long UsedCapacity
97          {
98              [MethodImpl(MethodImplOptions.AggressiveInlining)]
99              get
100             {
101                 Ensure.Always.NotDisposed(this);
102                 return _usedCapacity;
103             }
104             [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
            set
            {
                Ensure.Always.NotDisposed(this);
                if (value != _usedCapacity)
                {
                    Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
                    _usedCapacity = value;
                }
            }
        }

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override bool AllowMultipleDisposeCalls
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => true;
        }

        #endregion

        #region Methods

        /// <summary>
        /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
        ↪  property.</para>
        /// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
        /// </summary>
        /// <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>
        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
        ↪  newReservedCapacity);

        /// <summary>
        /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
        /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
        /// </summary>
        /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
        ↪  блок памяти.</para></param>
        /// <param name="usedCapacity"><para>The used capacity of the memory block in
        ↪  bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);

        #endregion

        #region DisposableBase Methods

        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
                if (pointer != IntPtr.Zero)
                {
                    DisposePointer(pointer, _usedCapacity);
                }
            }
        }

        #endregion
    }
}
```

1.11 ./csharp/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 ./csharp/Platform.Memory.Tests/HeapResizableDirectMemoryTests.cs

```
using Xunit;

namespace Platform.Memory.Tests
{
    /// <summary>
    /// <para>
    /// Represents the heap resizable direct memory tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe class HeapResizableDirectMemoryTests
    {
        /// <summary>
        /// <para>
        /// Tests that correct memory reallocation test.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
19          [Fact]
20          public void CorrectMemoryReallocationTest()
21          {
22              using var heapMemory = new HeapResizableDirectMemory();
23              var value1 = GetLastByte(heapMemory);
24              heapMemory.ReservedCapacity *= 2;
25              var value2 = GetLastByte(heapMemory);
26              Assert.Equal(value1, value2);
27              Assert.Equal(0, value1);
28          }
29          private static byte GetLastByte(HeapResizableDirectMemory heapMemory)
30          {
31              var pointer1 = (void*)heapMemory.Pointer;
32              return *((byte*)pointer1 + heapMemory.ReservedCapacity - 1);
33          }
34      }
35  }
```

Index