

LinksPlatform's Platform.Memory Class Library

1.1 ./Platform.Memory/ArrayMemory.cs

```
1 namespace Platform.Memory
2 {
3     /// <summary>
4     /// <para>Represents a memory block with access via indexer.</para>
5     /// <para>Представляет блок памяти с доступом через индекатор.</para>
6     /// </summary>
7     /// <typeparam name="TElement"><para>Element type.</para><para>Тип
8     ↪ элемента.</para></typeparam>
9     public class ArrayMemory<TElement> : IArrayMemory<TElement>
10     {
11         #region Fields
12
13         private readonly TElement[] _array;
14
15         #endregion
16
17         #region Properties
18
19         /// <inheritdoc/>
20         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
21         ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
22         public long Size => _array.Length;
23
24         /// <inheritdoc/>
25         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
26         ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
27         public TElement this[long index]
28         {
29             get => _array[index];
30             set => _array[index] = value;
31         }
32
33         #endregion
34
35         #region Constructors
36
37         /// <summary>
38         /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}"/>
39         ↪ class.</para>
40         /// <para>Инициализирует новый экземпляр класса <see
41         ↪ cref="ArrayMemory{TElement}"/>.</para>
42         /// </summary>
43         /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
44         public ArrayMemory(long size) => _array = new TElement[size];
45
46         #endregion
47     }
48 }
```

1.2 ./Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs

```
1 using System;
2 using Platform.Disposables;
3 using Platform.Exceptions;
4 using Platform.Unsafe;
5
6 namespace Platform.Memory
7 {
8     /// <summary>
9     /// <para>Represents adapter to a memory block with access via indexer.</para>
10    /// <para>Представляет адаптер к блоку памяти с доступом через индекатор.</para>
11    /// </summary>
12    /// <typeparam name="TElement"><para>Element type.</para><para>Тип
13    ↪ элемента.</para></typeparam>
14    public class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
15    ↪ IArrayMemory<TElement>, IDirectMemory
16    where TElement : struct
17    {
18        #region Fields
19
20        private readonly IDirectMemory _memory;
21
22        #endregion
23
24        #region Properties
25
26        /// <inheritdoc/>
27        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
28        ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
```

```

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

```

1.3 ./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>
13     public class FileArrayMemory<TElement> : DisposableBase, IArrayMemory<TElement> //-V3073

```

```

14     where TElement : struct
15 {
16     #region Fields
17
18     private readonly string _address;
19     private readonly FileStream _file;
20
21     #endregion
22
23     #region Properties
24
25     /// <inheritdoc/>
26     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
27     ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
28     public long Size => _file.Length;
29
30     /// <inheritdoc/>
31     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
32     ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
33     public TElement this[long index]
34     {
35         get
36         {
37             _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
38             return _file.ReadOrDefault<TElement>();
39         }
40         set
41         {
42             _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
43             _file.Write(value);
44         }
45     }
46
47     #endregion
48
49     #region DisposableBase Properties
50
51     /// <inheritdoc/>
52     protected override string ObjectName => $"File stored memory block at '{_address}'
53     ↪ path.";
54
55     #endregion
56
57     #region Constructors
58
59     /// <summary>
60     /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}" />
61     ↪ class.</para>
62     /// <para>Инициализирует новый экземпляр класса <see
63     ↪ cref="FileArrayMemory{TElement}" />.</para>
64     /// </summary>
65     /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
66     public FileArrayMemory(string path)
67     {
68         _address = path;
69         _file = File.Open(path, FileMode.OpenOrCreate, FileAccess.ReadWrite);
70     }
71
72     #endregion
73
74     #region DisposableBase Methods
75
76     /// <inheritdoc/>
77     protected override void Dispose(bool manual, bool wasDisposed)
78     {
79         if(!wasDisposed)
80         {
81             _file.DisposeIfPossible();
82         }
83     }
84
85     #endregion
86 }
87
88 }

```

1.4 ./Platform.Memory/FileMappedResizableDirectMemory.cs

```

1 using System;
2 using System.IO;
3 using System.IO.MemoryMappedFiles;
4 using Platform.Disposables;

```

```

5 using Platform.Exceptions;
6 using Platform.Collections;
7 using Platform.IO;
8
9 namespace Platform.Memory
10 {
11     /// <summary>
12     /// <para>Represents a memory block stored as a file on disk.</para>
13     /// <para>Представляет блок памяти, хранящийся в виде файла на диске.</para>
14     /// </summary>
15     public unsafe class FileMappedResizableDirectMemory : ResizableDirectMemoryBase
16     {
17         #region Fields
18
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 => $"File stored memory block at '{Path}' path.";
34
35         #endregion
36
37         #region Constructors
38
39         /// <summary>
40         /// <para>Initializes a new instance of the <see
41         ↪ cref="FileMappedResizableDirectMemory"/> class.</para>
42         /// <para>Инициализирует новый экземпляр класса <see
43         ↪ cref="FileMappedResizableDirectMemory"/>.</para>
44         /// </summary>
45         /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
46         /// <param name="minimumReservedCapacity"><para>Minimum file size in
47         ↪ bytes.</para><para>Минимальный размер файла в байтах.</para></param>
48         public FileMappedResizableDirectMemory(string path, long minimumReservedCapacity)
49         {
50             Ensure.Always.ArgumentNotEmptyAndNotWhiteSpace(path, nameof(path));
51             if (minimumReservedCapacity < MinimumCapacity)
52             {
53                 minimumReservedCapacity = MinimumCapacity;
54             }
55             Path = path;
56             var size = FileHelpers.GetSize(Path);
57             ReservedCapacity = size > minimumReservedCapacity ? ((size /
58             ↪ minimumReservedCapacity) + 1) * minimumReservedCapacity :
59             ↪ minimumReservedCapacity;
60             UsedCapacity = size;
61         }
62
63         /// <summary>
64         /// <para>Initializes a new instance of the <see
65         ↪ cref="FileMappedResizableDirectMemory"/> class.</para>
66         /// <para>Инициализирует новый экземпляр класса <see
67         ↪ cref="FileMappedResizableDirectMemory"/>.</para>
68         /// </summary>
69         /// <param name="address"><para>An path to file.</para><para>Путь к файлу.</para></param>
70         public FileMappedResizableDirectMemory(string address) : this(address, MinimumCapacity)
71         ↪ { }
72
73         #endregion
74
75         #region Methods
76
77         private void MapFile(long capacity)
78         {
79             if (Pointer != IntPtr.Zero)
80             {
81                 return;
82             }
83             _file = MemoryMappedFile.CreateFromFile(Path, FileMode.Open, mapName: null,
84             ↪ capacity, MemoryMappedFileAccess.ReadWrite);
85         }
86     }
87 }

```

```

76     _accessor = _file.CreateViewAccessor();
77     byte* pointer = null;
78     _accessor.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
79     Pointer = new IntPtr(pointer);
80 }
81
82 private void UnmapFile()
83 {
84     if (UnmapFile(Pointer))
85     {
86         Pointer = IntPtr.Zero;
87     }
88 }
89
90 private bool UnmapFile(IntPtr pointer)
91 {
92     if (pointer == IntPtr.Zero)
93     {
94         return false;
95     }
96     if (_accessor != null)
97     {
98         _accessor.SafeMemoryMappedViewHandle.ReleasePointer();
99         Disposable.TryDisposeAndResetToDefault(ref _accessor);
100     }
101     Disposable.TryDisposeAndResetToDefault(ref _file);
102     return true;
103 }
104
105 #endregion
106
107 #region ResizableDirectMemoryBase Methods
108
109 /// <inheritdoc>
110 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
111   ↳ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReservedCapacityChanged(System.Int64,System.Int64)"]/*' />
112   ↳ edCapacityChanged(System.Int64,System.Int64)"/>
113 protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
114   ↳ newReservedCapacity)
115 {
116     UnmapFile();
117     FileHelpers.SetSize(Path, newReservedCapacity);
118     MapFile(newReservedCapacity);
119 }
120
121 /// <inheritdoc>
122 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
123   ↳ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposePointer(System.IntPtr,System.Int64)"]/*' />
124   ↳ ointer(System.IntPtr,System.Int64)"/>
125 protected override void DisposePointer(IntPtr pointer, long usedCapacity)
126 {
127     if (UnmapFile(pointer))
128     {
129         FileHelpers.SetSize(Path, usedCapacity);
130     }
131 }
132
133 #endregion
134 }
135 }

```

1.5 ./Platform.Memory/HeapResizableDirectMemory.cs

```

1 using Platform.Unsafe;
2 using System;
3 using System.Runtime.InteropServices;
4
5 namespace Platform.Memory
6 {
7     /// <summary>
8     /// <para>Represents a memory block allocated in Heap.</para>
9     /// <para>Представляет блок памяти, выделенный в "куче".</para>
10    /// </summary>
11    public unsafe class HeapResizableDirectMemory : ResizableDirectMemoryBase
12    {
13        #region DisposableBase Properties
14
15        /// <inheritdoc>
16        protected override string ObjectName => $"Heap stored memory block at {Pointer}
17   ↳ address.";

```

```

17 #endregion
18
19 #region Constructors
20
21 /// <summary>
22 /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
23   ↳ class.</para>
24 /// <para>Инициализирует новый экземпляр класса <see
25   ↳ cref="HeapResizableDirectMemory"/>.</para>
26 /// </summary>
27 /// <param name="minimumReservedCapacity"><para>Minimum file size in
28   ↳ bytes.</para><para>Минимальный размер файла в байтах.</para></param>
29 public HeapResizableDirectMemory(long minimumReservedCapacity)
30 {
31     if (minimumReservedCapacity < MinimumCapacity)
32     {
33         minimumReservedCapacity = MinimumCapacity;
34     }
35     ReservedCapacity = minimumReservedCapacity;
36     UsedCapacity = 0;
37 }
38
39 /// <summary>
40 /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
41   ↳ class.</para>
42 /// <para>Инициализирует новый экземпляр класса <see
43   ↳ cref="HeapResizableDirectMemory"/>.</para>
44 /// </summary>
45 public HeapResizableDirectMemory() : this(MinimumCapacity) { }
46
47 #endregion
48
49 #region ResizableDirectMemoryBase Methods
50
51 /// <inheritdoc/>
52 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
53   ↳ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
54   ↳ ointer(System.IntPtr,System.Int64)"]/*' />
55 protected override void DisposePointer(IntPtr pointer, long usedCapacity) =>
56   ↳ Marshal.FreeHGlobal(pointer);
57
58 /// <inheritdoc/>
59 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
60   ↳ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
61   ↳ edCapacityChanged(System.Int64,System.Int64)"]/*' />
62 protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
63   ↳ newReservedCapacity)
64 {
65     if (Pointer == IntPtr.Zero)
66     {
67         Pointer = Marshal.AllocHGlobal(new IntPtr(newReservedCapacity));
68         MemoryBlock.Zero((void*)Pointer, newReservedCapacity);
69     }
70     else
71     {
72         Pointer = Marshal.ReAllocHGlobal(Pointer, new IntPtr(newReservedCapacity));
73         var pointer = (byte*)Pointer + oldReservedCapacity;
74         MemoryBlock.Zero(pointer, newReservedCapacity - oldReservedCapacity);
75     }
76 }
77
78 #endregion
79
80 }
81
82 }

```

1.6 ./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 индекатор.</para>
6     /// </summary>
7     /// <typeparam name="TElement"><para>Element type.</para><para>Тип
8     ↳ элемента.</para></typeparam>
9     public interface IArrayMemory<TElement> : IMemory
10     {
11         /// <summary>

```

```

11     /// <para>Gets or sets the element at the specified index.</para>
12     /// <para>Возвращает или устанавливает элемент по указанному индексу.</para>
13     /// </summary>
14     /// <param name="index"><para>The index of the element to get or set.</para><para>Индекс
    ↪ элемента, который нужно получить или установить.</para></param>
15     TElement this[long index] { get; set; }
16 }
17 }

```

1.7 ./Platform.Memory/IDirectMemory.cs

```

1 using System;
2
3 namespace Platform.Memory
4 {
5     /// <summary>
6     /// <para>Represents a memory block interface with direct access (via unmanaged
    ↪ pointers).</para>
7     /// <para>Представляет интерфейс блока памяти с прямым доступом (через неуправляемые
    ↪ указатели).</para>
8     /// </summary>
9     public interface IDirectMemory : IMemory, IDisposable
10     {
11         /// <summary>
12         /// <para>Gets the pointer to the beginning of this memory block.</para>
13         /// <para>Возвращает указатель на начало блока памяти.</para>
14         /// </summary>
15         IntPtr Pointer { get; }
16     }
17 }

```

1.8 ./Platform.Memory/IMemory.cs

```

1 namespace Platform.Memory
2 {
3     /// <summary>
4     /// <para>Represents a memory block interface with size in bytes.</para>
5     /// <para>Представляет интерфейс блока памяти с размером в байтах.</para>
6     /// </summary>
7     public interface IMemory
8     {
9         /// <summary>
10        /// <para>Gets the size in bytes of this memory block.</para>
11        /// <para>Возвращает размер блока памяти в байтах.</para>
12        /// </summary>
13        long Size { get; }
14    }
15 }

```

1.9 ./Platform.Memory/IResizableDirectMemory.cs

```

1 namespace Platform.Memory
2 {
3     /// <summary>
4     /// <para>Represents a resizable memory block interface with direct access (via unmanaged
    ↪ pointers).</para>
5     /// <para>Представляет интерфейс блока памяти с изменяемым размером и прямым доступом (через
    ↪ неуправляемые указатели).</para>
6     /// </summary>
7     public interface IResizableDirectMemory : IDirectMemory
8     {
9         /// <summary>
10        /// <para>Gets or sets the reserved capacity in bytes of this memory block.</para>
11        /// <para>Возвращает или устанавливает зарезервированный размер блока памяти в
    ↪ байтах.</para>
12        /// </summary>
13        /// <remarks>
14        /// <para>
15        /// If less than zero the value is replaced with zero.
16        /// Cannot be less than the used capacity of this memory block.
17        /// </para>
18        /// <para>
19        /// Если меньше нуля, значение заменяется на ноль.
20        /// Не может быть меньше используемой емкости блока памяти.
21        /// </para>
22        /// </remarks>
23        long ReservedCapacity { get; set; }
24
25        /// <summary>
26        /// <para>Gets or sets the used capacity in bytes of this memory block.</para>

```

```

27     /// <para>Возвращает или устанавливает используемый размер в блоке памяти (в
    → байтах).</para>
28     /// </summary>
29     /// <remarks>
30     /// <para>
31     /// If less then zero the value is replaced with zero.
32     /// Cannot be greater than the reserved capacity of this memory block.
33     /// </para>
34     /// <para>
35     /// 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.
36     /// </para>
37     /// <para>
38     /// Если меньше нуля, значение заменяется на ноль.
39     /// Не может быть больше, чем зарезервированная емкость этого блока памяти.
40     /// </para>
41     /// <para>
42     /// Рекомендуется уменьшать фактический размер блока памяти до используемого размера
    → (указанного в этом свойстве) после завершения использования блока памяти.
43     /// </para>
44     /// </remarks>
45     long UsedCapacity { get; set; }
46 }
47 }

```

1.10 ./Platform.Memory/ResizableDirectMemoryBase.cs

```

1  using System;
2  using System.Threading;
3  using Platform.Exceptions;
4  using Platform.Disposables;
5  using Platform.Ranges;
6
7  namespace Platform.Memory
8  {
9      /// <summary>
10     /// <para>Provides a base implementation for the resizable memory block with direct access
    → (via unmanaged pointers).</para>
11     /// <para>Предоставляет базовую реализацию для блока памяти с изменяемым размером и прямым
    → доступом (через неуправляемые указатели).</para>
12     /// </summary>
13     public abstract class ResizableDirectMemoryBase : DisposableBase, IResizableDirectMemory
14     {
15         #region Constants
16
17         /// <summary>
18         /// <para>Gets minimum capacity in bytes.</para>
19         /// <para>Возвращает минимальную емкость в байтах.</para>
20         /// </summary>
21         public static readonly long MinimumCapacity = 4096;
22
23         #endregion
24
25         #region Fields
26
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             get
41             {
42                 Ensure.Always.NotDisposed(this);
43                 return UsedCapacity;
44             }
45         }
46
47         /// <inheritdoc>
48         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
    → path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*' />

```



```

49     /// <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 #endregion
116
117 #region Methods
118
119 /// <summary>
120 /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
121   → 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
125   → block in bytes.</para><para>Старая зарезервированная емкость блока памяти в
126   → байтах.</para></param>
127 /// <param name="newReservedCapacity"><para>The new reserved capacity of the memory
128   → block in bytes.</para><para>Новая зарезервированная емкость блока памяти в
129   → байтах.</para></param>
130 protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
131   → newReservedCapacity);
132
133 /// <summary>
134 /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
135 /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
136 /// </summary>
137 /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
138   → блок памяти.</para></param>
139 /// <param name="usedCapacity"><para>The used capacity of the memory block in
140   → bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
141 protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
142
143 #endregion
144
145 #region DisposableBase Methods
146
147 /// <inheritdoc>
148 protected override void Dispose(bool manual, bool wasDisposed)
149 {
150     if (!wasDisposed)
151     {
152         var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
153         if (pointer != IntPtr.Zero)
154         {
155             DisposePointer(pointer, _usedCapacity);
156         }
157     }
158 }
159
160 #endregion
161 }
162
163 }
```

1.11 ./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
15   → '{Path}' path.";
16
17         #endregion
18
19         #region Constructors
20
21         /// <summary>
22         /// <para>Initializes a new instance of the <see
23   → cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
24         /// <para>Инициализирует новый экземпляр класса <see
25   → cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
26         /// </summary>
27         /// <param name="minimumReservedCapacity"><para>Minimum file size in
28   → bytes.</para><para>Минимальный размер файла в байтах.</para></param>
29     }
```

```

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 }

```

1.12 ./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                  var value1 = GetLastByte(heapMemory);
13                  heapMemory.ReservedCapacity *= 2;
14                  var value2 = GetLastByte(heapMemory);
15                  Assert.Equal(value1, value2);
16                  Assert.Equal(0, value1);
17              }
18          }
19
20          private static byte GetLastByte(HeapResizableDirectMemory heapMemory)
21          {
22              var pointer1 = (void*)heapMemory.Pointer;
23              return *((byte*)pointer1 + heapMemory.ReservedCapacity - 1);
24          }
25      }
26  }

```

Index

- ./Platform.Memory.Tests/HeapResizableDirectMemoryTests.cs, 11
- ./Platform.Memory/ArrayMemory.cs, 1
- ./Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs, 1
- ./Platform.Memory/FileArrayMemory.cs, 2
- ./Platform.Memory/FileMappedResizableDirectMemory.cs, 3
- ./Platform.Memory/HeapResizableDirectMemory.cs, 5
- ./Platform.Memory/IArrayMemory.cs, 6
- ./Platform.Memory/IDirectMemory.cs, 7
- ./Platform.Memory/IMemory.cs, 7
- ./Platform.Memory/IResizableDirectMemory.cs, 7
- ./Platform.Memory/ResizableDirectMemoryBase.cs, 8
- ./Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs, 10