

LinksPlatform's Platform.Memory Class Library

./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/member[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
26         public TElement this[long index]
27         {
28             get => _array[index];
29             set => _array[index] = value;
30         }
31
32         #endregion
33
34         #region Constructors
35
36         /// <summary>
37         /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}" />
38         ↪ class.</para>
39         /// <para>Инициализирует новый экземпляр класса <see
40         ↪ cref="ArrayMemory{TElement}" />.</para>
41         /// </summary>
42         /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
43         public ArrayMemory(long size) => _array = new TElement[size];
44
45         #endregion
46     }
47 }
```

./Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs

```

1  using System;
2  using Platform.Disposables;
3  using Platform.Exceptions;
4
5  namespace Platform.Memory
6  {
7      /// <summary>
8      /// <para>Represents adapter to a memory block with access via indexer.</para>
9      /// <para>Представляет адаптер к блоку памяти с доступом через индекатор.</para>
10     /// </summary>
11     /// <typeparam name="TElement"><para>Element type.</para><para>Тип
12     ↪ элемента.</para></typeparam>
13     public unsafe class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
14     ↪ IArrayMemory<TElement>, IDirectMemory
15     where TElement : struct
16     {
17         #region Fields
18
19         private readonly IDirectMemory _memory;
20
21         #endregion
22
23         #region Properties
24
25         /// <inheritdoc/>
26         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
27         ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
28         public long Size => _memory.Size;
29     }
```

```

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>

```

```

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 }

```

./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         }
84     }
85 }

```

```

75         _file = MemoryMappedFile.CreateFromFile(Path, FileMode.Open, mapName: null,
76         ↪ capacity, MemoryMappedFileAccess.ReadWrite);
77         _accessor = _file.CreateViewAccessor();
78         byte* pointer = null;
79         _accessor.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
80         Pointer = new IntPtr(pointer);
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.OnReserv
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.DisposeP
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 }

```

./Platform.Memory/HeapResizableDirectMemory.cs

```

1  using System;
2  using System.Runtime.InteropServices;
3
4  namespace Platform.Memory
5  {
6      /// <summary>
7      /// <para>Represents a memory block allocated in Heap.</para>
8      /// <para>Представляет блок памяти, выделенный в "куче".</para>
9      /// </summary>
10     public unsafe class HeapResizableDirectMemory : ResizableDirectMemoryBase
11     {
12         #region DisposableBase Properties
13
14         /// <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 индекатор.</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>

```

```

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 }

```

./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 }

```

./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 }

```

./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 then 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 }

```

./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_
66   ↳ ber[@name="P:Platform.Memory.IResizableDirectMemory.ReservedCapacity"]/*' />
67 /// <exception cref="ObjectDisposedException"><para>The memory block is
68   ↳ disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
69 /// <exception cref="ArgumentOutOfRangeException"><para>Attempted to set the reserved
70   ↳ capacity to a value that is less than the used capacity.</para><para>Была выполнена
71   ↳ попытка установить зарезервированную емкость на значение, которое меньше
72   ↳ используемой емкости.</para></exception>
73 public long ReservedCapacity
74 {
75     get
76     {
77         Ensure.Always.NotDisposed(this);
78         return _reservedCapacity;
79     }
80     set
81     {
82         Ensure.Always.NotDisposed(this);
83         if (value != _reservedCapacity)
84         {
85             Ensure.Always.ArgumentInRange(value, new Range<long>(_usedCapacity,
86   ↳ long.MaxValue));
87             OnReservedCapacityChanged(_reservedCapacity, value);
88             _reservedCapacity = value;
89         }
90     }
91 }
92
93 /// <inheritdoc>
94 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem_
95   ↳ ber[@name="P:Platform.Memory.IResizableDirectMemory.UsedCapacity"]/*' />
96 /// <exception cref="ObjectDisposedException"><para>The memory block is
97   ↳ disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
98 /// <exception cref="ArgumentOutOfRangeException"><para>Attempted to set the used
99   ↳ capacity to a value that is greater than the reserved capacity or less than
100   ↳ zero.</para><para>Была выполнена попытка установить используемую емкость на
101   ↳ значение, которое больше, чем зарезервированная емкость или меньше
102   ↳ нуля.</para></exception>
103 public long UsedCapacity
104 {
105     get
106     {
107         Ensure.Always.NotDisposed(this);
108         return _usedCapacity;
109     }
110     set
111     {
112         Ensure.Always.NotDisposed(this);
113         if (value != _usedCapacity)
114         {
115             Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
116             _usedCapacity = value;
117         }
118     }
119 }
120
121 #endregion
122
123 #region DisposableBase Properties
124
125 /// <inheritdoc>
126 protected override bool AllowMultipleDisposeCalls => true;
127
128 #endregion

```

```

117 #region Methods
118
119
120 /// <summary>
121 /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
122   ↳ property.</para>
123 /// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
124 /// </summary>
125 /// <param name="oldReservedCapacity"><para>The old reserved capacity of the memory
126   ↳ block in bytes.</para><para>Старая зарезервированная емкость блока памяти в
127   ↳ байтах.</para></param>
128 /// <param name="newReservedCapacity"><para>The new reserved capacity of the memory
129   ↳ block in bytes.</para><para>Новая зарезервированная емкость блока памяти в
130   ↳ байтах.</para></param>
131 protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
132   ↳ newReservedCapacity);
133
134 /// <summary>
135 /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
136 /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
137 /// </summary>
138 /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
139   ↳ блок памяти.</para></param>
140 /// <param name="usedCapacity"><para>The used capacity of the memory block in
141   ↳ bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
142 protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
143
144 #endregion
145
146 #region DisposableBase Methods
147
148 /// <inheritdoc/>
149 protected override void Dispose(bool manual, bool wasDisposed)
150 {
151     if (!wasDisposed)
152     {
153         var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
154         if (pointer != IntPtr.Zero)
155         {
156             DisposePointer(pointer, _usedCapacity);
157         }
158     }
159 }
160
161 #endregion
162 }
163
164 }

```

./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         public TemporaryFileMappedResizableDirectMemory(long minimumReservedCapacity) :
30   ↳ base(System.IO.Path.GetTempFileName(), minimumReservedCapacity) { }
31     }
32 }

```

```

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 = *((byte*)pointer1 + heapMemory.ReservedCapacity - 1);
14
15                     heapMemory.ReservedCapacity *= 2;
16
17                     void* pointer2 = (void*)heapMemory.Pointer;
18                     var value2 = *((byte*)pointer2 + heapMemory.ReservedCapacity - 1);
19
20                     Assert.Equal(value1, value2);
21                     Assert.Equal(0, value1);
22                 }
23             }
24         }
25     }

```

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