

LinksPlatform's Platform.Memory Class Library

1.1 ./csharp/Platform.Memory/ArrayMemory.cs

```
1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Memory
4 {
5     /// <summary>
6     /// <para>Represents a memory block with access via indexer.</para>
7     /// <para>Представляет блок памяти с доступом через индекатор.</para>
8     /// </summary>
9     /// <typeparam name="TElement"><para>Element type.</para><para>Тип
10     ↪ элемента.</para></typeparam>
11 public class ArrayMemory<TElement> : IArrayMemory<TElement>
12 {
13     #region Fields
14     private readonly TElement[] _array;
15
16     #endregion
17
18     #region Properties
19
20     /// <inheritdoc/>
21     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
22     ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
23 public long Size
24 {
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     get => _array.Length;
27 }
28
29     /// <inheritdoc/>
30     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
31     ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
32 public TElement this[long index]
33 {
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     get => _array[index];
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     set => _array[index] = value;
38 }
39
40     #endregion
41
42     #region Constructors
43
44     /// <summary>
45     /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}" />
46     ↪ class.</para>
47     /// <para>Инициализирует новый экземпляр класса <see
48     ↪ cref="ArrayMemory{TElement}" />.</para>
49     /// </summary>
50     /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     public ArrayMemory(long size) => _array = new TElement[size];
53
54     #endregion
55 }
56 }
```

1.2 ./csharp/Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Disposables;
4 using Platform.Exceptions;
5 using Platform.Unsafe;
6
7 namespace Platform.Memory
8 {
9     /// <summary>
10     /// <para>Represents adapter to a memory block with access via indexer.</para>
11     /// <para>Представляет адаптер к блоку памяти с доступом через индекатор.</para>
12     /// </summary>
13     /// <typeparam name="TElement"><para>Element type.</para><para>Тип
14     ↪ элемента.</para></typeparam>
15 public class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
16     ↪ IArrayMemory<TElement>, IDirectMemory
17     where TElement : struct
18 {
19     #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'
26   ↳ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
27 public long Size
28 {
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     get => _memory.Size;
31 }
32
33 /// <inheritdoc/>
34 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
35   ↳ path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*' />
36 public IntPtr Pointer
37 {
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     get => _memory.Pointer;
40 }
41
42 /// <inheritdoc/>
43 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
44   ↳ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
45 public TElement this[long index]
46 {
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     get => Pointer.ReadElementValue<TElement>(index);
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     set => Pointer.WriteElementValue(index, value);
51 }
52
53 #endregion
54
55 #region DisposableBase Properties
56
57 /// <inheritdoc/>
58 protected override string ObjectName
59 {
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     get => $"Array as memory block at '{Pointer}' address.";
62 }
63
64 #endregion
65
66 #region Constructors
67
68 /// <summary>
69 /// <para>Initializes a new instance of the <see
70   ↳ cref="DirectMemoryAsArrayMemoryAdapter{TElement}" /> class.</para>
71 /// <para>Инициализирует новый экземпляр класса <see
72   ↳ cref="DirectMemoryAsArrayMemoryAdapter{TElement}" />.</para>
73 /// </summary>
74 /// <param name="memory"><para>An object implementing <see cref="IDirectMemory"/>
75   ↳ interface.</para><para>Объект, реализующий интерфейс <see
76   ↳ cref="IDirectMemory"/>.</para></param>
77 [MethodImpl(MethodImplOptions.AggressiveInlining)]
78 public DirectMemoryAsArrayMemoryAdapter(IDirectMemory memory)
79 {
80     Ensure.Always.ArgumentNotNull(memory, nameof(memory));
81     Ensure.Always.ArgumentMeetsCriteria(memory, m => (m.Size % Structure<TElement>.Size)
82   ↳ == 0, nameof(memory), "Memory is not aligned to element size.");
83     _memory = memory;
84 }
85
86 #endregion
87
88 #region DisposableBase Methods
89
90 /// <inheritdoc/>
91 [MethodImpl(MethodImplOptions.AggressiveInlining)]
92 protected override void Dispose(bool manual, bool wasDisposed)
93 {
94     if (!wasDisposed)
95     {
96         _memory.DisposeIfPossible();
97     }
98 }
99

```

```

89     }
90 }
91
92 #endregion
93 }
94 }

```

1.3 ./csharp/Platform.Memory/FileArrayMemory.cs

```

1  using System.IO;
2  using System.Runtime.CompilerServices;
3  using Platform.Disposables;
4  using Platform.Unsafe;
5  using Platform.IO;
6
7  namespace Platform.Memory
8  {
9      /// <summary>
10     /// <para>Represents a memory block with access via indexer and stored as file on
11     ↪ disk.</para>
12     /// <para>Представляет блок памяти с доступом через индексатор и хранящийся в виде файла на
13     ↪ диске.</para>
14     /// </summary>
15     /// <typeparam name="TElement"><para>Element type.</para><para>Тип
16     ↪ элемента.</para></typeparam>
17     public class FileArrayMemory<TElement> : DisposableBase, IArrayMemory<TElement> //-V3073
18     where TElement : struct
19     {
20         #region Fields
21         private readonly FileStream _file;
22
23         #endregion
24
25         #region Properties
26
27         /// <inheritdoc/>
28         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
29         ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
30         public long Size
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get => _file.Length;
34         }
35
36         /// <inheritdoc/>
37         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
38         ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
39         public TElement this[long index]
40         {
41             [MethodImpl(MethodImplOptions.AggressiveInlining)]
42             get
43             {
44                 _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
45                 return _file.ReadOrDefault<TElement>();
46             }
47             [MethodImpl(MethodImplOptions.AggressiveInlining)]
48             set
49             {
50                 _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
51                 _file.Write(value);
52             }
53         }
54
55         #endregion
56
57         #region DisposableBase Properties
58
59         /// <inheritdoc/>
60         protected override string ObjectName
61         {
62             [MethodImpl(MethodImplOptions.AggressiveInlining)]
63             get => $"File stored memory block at '{_file.Name}' path.";
64         }
65
66         #endregion
67
68         #region Constructors
69
70         /// <summary>
71         /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}">
72         ↪ 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'
117     ↪ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
118     ↪ edCapacityChanged(System.Int64,System.Int64)"]/*' />
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
121     ↪ newReservedCapacity)
122     {
123         UnmapFile();
124         FileHelpers.SetSize(Path, newReservedCapacity);
125         MapFile(newReservedCapacity);
126     }
127
128     /// <inheritdoc>
129     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
130     ↪ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
131     ↪ ointer(System.IntPtr,System.Int64)"]/*' />
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     protected override void DisposePointer(IntPtr pointer, long usedCapacity)
134     {
135         if (UnmapFile(pointer))
136         {
137             FileHelpers.SetSize(Path, usedCapacity);
138         }
139     }
140 }
141 #endregion
142 }
143 }

```

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"/>
29         ↪ class.</para>
30         /// <para>Инициализирует новый экземпляр класса <see
31         ↪ cref="HeapResizableDirectMemory"/>.</para>
32         /// </summary>
33         /// <param name="minimumReservedCapacity"><para>Minimum file size in
34         ↪ bytes.</para><para>Минимальный размер файла в байтах.</para></param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public HeapResizableDirectMemory(long minimumReservedCapacity)
37         {
38             if (minimumReservedCapacity < MinimumCapacity)
39             {
40                 minimumReservedCapacity = MinimumCapacity;
41             }
42             ReservedCapacity = minimumReservedCapacity;
43             UsedCapacity = 0;
44         }
45
46         /// <summary>
47         /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
48         ↪ 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 {

```

```

6     /// <summary>
7     /// <para>Represents a memory block interface with direct access (via unmanaged
    ⇨ pointers).</para>
8     /// <para>Представляет интерфейс блока памяти с прямым доступом (через неуправляемые
    ⇨ указатели).</para>
9     /// </summary>
10    public interface IDirectMemory : IMemory, IDisposable
11    {
12        /// <summary>
13        /// <para>Gets the pointer to the beginning of this memory block.</para>
14        /// <para>Возвращает указатель на начало блока памяти.</para>
15        /// </summary>
16        IntPtr Pointer
17        {
18            [MethodImpl(MethodImplOptions.AggressiveInlining)]
19            get;
20        }
21    }
22 }

```

1.8 ./csharp/Platform.Memory/IMemory.cs

```

1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Memory
4 {
5     /// <summary>
6     /// <para>Represents a memory block interface with size in bytes.</para>
7     /// <para>Представляет интерфейс блока памяти с размером в байтах.</para>
8     /// </summary>
9     public interface IMemory
10    {
11        /// <summary>
12        /// <para>Gets the size in bytes of this memory block.</para>
13        /// <para>Возвращает размер блока памяти в байтах.</para>
14        /// </summary>
15        long Size
16        {
17            [MethodImpl(MethodImplOptions.AggressiveInlining)]
18            get;
19        }
20    }
21 }

```

1.9 ./csharp/Platform.Memory/IResizableDirectMemory.cs

```

1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Memory
4 {
5     /// <summary>
6     /// <para>Represents a resizable memory block interface with direct access (via unmanaged
    ⇨ pointers).</para>
7     /// <para>Представляет интерфейс блока памяти с изменяемым размером и прямым доступом (через
    ⇨ неуправляемые указатели).</para>
8     /// </summary>
9     public interface IResizableDirectMemory : IDirectMemory
10    {
11        /// <summary>
12        /// <para>Gets or sets the reserved capacity in bytes of this memory block.</para>
13        /// <para>Возвращает или устанавливает зарезервированный размер блока памяти в
    ⇨ байтах.</para>
14        /// </summary>
15        /// <remarks>
16        /// <para>
17        /// If less then zero the value is replaced with zero.
18        /// Cannot be less than the used capacity of this memory block.
19        /// </para>
20        /// <para>
21        /// Если меньше нуля, значение заменяется на ноль.
22        /// Не может быть меньше используемой емкости блока памяти.
23        /// </para>
24        /// </remarks>
25        long ReservedCapacity
26        {
27            [MethodImpl(MethodImplOptions.AggressiveInlining)]
28            get;
29            [MethodImpl(MethodImplOptions.AggressiveInlining)]
30            set;
31        }
32    }
33 }

```



```

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'
50   → path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*' />
51 /// <exception cref="ObjectDisposedException"><para>The memory block is
52   → disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
53 public IntPtr Pointer
54 {
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     get
57     {
58         Ensure.Always.NotDisposed(this);
59         return _pointer;
60     }
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     protected set
63     {
64         Ensure.Always.NotDisposed(this);
65         _pointer = value;
66     }
67 }
68
69 /// <inheritdoc>
70 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem_
71   → ber[@name="P:Platform.Memory.IResizableDirectMemory.ReservedCapacity"]/*' />
72 /// <exception cref="ObjectDisposedException"><para>The memory block is
73   → disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
74 /// <exception cref="ArgumentOutOfRangeException"><para>Attempted to set the reserved
75   → capacity to a value that is less than the used capacity.</para><para>Была выполнена
76   → попытка установить зарезервированную емкость на значение, которое меньше
77   → используемой емкости.</para></exception>
78 public long ReservedCapacity
79 {
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     get
82     {
83         Ensure.Always.NotDisposed(this);
84         return _reservedCapacity;
85     }
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     set
88     {
89         Ensure.Always.NotDisposed(this);
90         if (value != _reservedCapacity)
91         {
92             Ensure.Always.ArgumentInRange(value, new Range<long>(_usedCapacity,
93   → long.MaxValue));
94             OnReservedCapacityChanged(_reservedCapacity, value);
95             _reservedCapacity = value;
96         }
97     }
98 }
99
100 /// <inheritdoc>
101 /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem_
102   → ber[@name="P:Platform.Memory.IResizableDirectMemory.UsedCapacity"]/*' />
103 /// <exception cref="ObjectDisposedException"><para>The memory block is
104   → disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
105 /// <exception cref="ArgumentOutOfRangeException"><para>Attempted to set the used
106   → capacity to a value that is greater than the reserved capacity or less than
107   → zero.</para><para>Была выполнена попытка установить используемую емкость на
108   → значение, которое больше, чем зарезервированная емкость или меньше
109   → нуля.</para></exception>
110 public long UsedCapacity
111 {
112     [MethodImpl(MethodImplOptions.AggressiveInlining)]
113     get
114     {
115         Ensure.Always.NotDisposed(this);
116         return _usedCapacity;
117     }
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

105         set
106         {
107             Ensure.Always.NotDisposed(this);
108             if (value != _usedCapacity)
109             {
110                 Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
111                 _usedCapacity = value;
112             }
113         }
114     }
115
116     #endregion
117
118     #region DisposableBase Properties
119
120     /// <inheritdoc>
121     protected override bool AllowMultipleDisposeCalls
122     {
123         [MethodImpl(MethodImplOptions.AggressiveInlining)]
124         get => true;
125     }
126
127     #endregion
128
129     #region Methods
130
131     /// <summary>
132     /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
133     ///     ↳ property.</para>
134     /// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
135     /// </summary>
136     /// <param name="oldReservedCapacity"><para>The old reserved capacity of the memory
137     ///     ↳ block in bytes.</para><para>Старая зарезервированная емкость блока памяти в
138     ///     ↳ байтах.</para></param>
139     /// <param name="newReservedCapacity"><para>The new reserved capacity of the memory
140     ///     ↳ block in bytes.</para><para>Новая зарезервированная емкость блока памяти в
141     ///     ↳ байтах.</para></param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
144     ↳ newReservedCapacity);
145
146     /// <summary>
147     /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
148     /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
149     /// </summary>
150     /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
151     ///     ↳ блок памяти.</para></param>
152     /// <param name="usedCapacity"><para>The used capacity of the memory block in
153     ///     ↳ bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
156
157     #endregion
158
159     #region DisposableBase Methods
160
161     /// <inheritdoc>
162     [MethodImpl(MethodImplOptions.AggressiveInlining)]
163     protected override void Dispose(bool manual, bool wasDisposed)
164     {
165         if (!wasDisposed)
166         {
167             var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
168             if (pointer != IntPtr.Zero)
169             {
170                 DisposePointer(pointer, _usedCapacity);
171             }
172         }
173     }
174
175     #endregion
176 }
177
178 }
```

1.11 ./csharp/Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs

```

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

```

5 {
6     /// <summary>
7     /// <para>Represents a memory block stored as a temporary file on disk.</para>
8     /// <para>Представляет блок памяти, хранящийся в виде временного файла на диске.</para>
9     /// </summary>
10    public class TemporaryFileMappedResizableDirectMemory : FileMappedResizableDirectMemory
11    {
12        #region DisposableBase Properties
13
14        /// <inheritdoc/>
15        protected override string ObjectName
16        {
17            [MethodImpl(MethodImplOptions.AggressiveInlining)]
18            get => $"Temporary file stored memory block at '{Path}' path.";
19        }
20
21        #endregion
22
23        #region Constructors
24
25        /// <summary>
26        /// <para>Initializes a new instance of the <see
27        ///     ↳ cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
28        /// <para>Инициализирует новый экземпляр класса <see
29        ///     ↳ cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
30        /// </summary>
31        /// <param name="minimumReservedCapacity"><para>Minimum file size in
32        ///     ↳ bytes.</para><para>Минимальный размер файла в байтах.</para></param>
33        [MethodImpl(MethodImplOptions.AggressiveInlining)]
34        public TemporaryFileMappedResizableDirectMemory(long minimumReservedCapacity) :
35        ↳ base(System.IO.Path.GetTempFileName(), minimumReservedCapacity) { }
36
37        /// <summary>
38        /// <para>Initializes a new instance of the <see
39        ///     ↳ cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
40        /// <para>Инициализирует новый экземпляр класса <see
41        ///     ↳ cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
42        /// </summary>
43        [MethodImpl(MethodImplOptions.AggressiveInlining)]
44        public TemporaryFileMappedResizableDirectMemory() : this(MinimumCapacity) { }
45
46        #endregion
47
48        #region DisposableBase Methods
49
50        /// <inheritdoc/>
51        [MethodImpl(MethodImplOptions.AggressiveInlining)]
52        protected override void Dispose(bool manual, bool wasDisposed)
53        {
54            base.Dispose(manual, wasDisposed);
55            if (!wasDisposed)
56            {
57                File.Delete(Path);
58            }
59        }
60
61        #endregion
62    }
63 }

```

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

```

1 using Xunit;
2
3 namespace Platform.Memory.Tests
4 {
5     /// <summary>
6     /// <para>
7     ///     Represents the heap resizable direct memory tests.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    public unsafe class HeapResizableDirectMemoryTests
12    {
13        /// <summary>
14        /// <para>
15        ///     Tests that correct memory reallocation test.
16        /// </para>
17        /// <para></para>
18        /// </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

- ./csharp/Platform.Memory.Tests/HeapResizableDirectMemoryTests.cs, 12
- ./csharp/Platform.Memory/ArrayMemory.cs, 1
- ./csharp/Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs, 1
- ./csharp/Platform.Memory/FileArrayMemory.cs, 3
- ./csharp/Platform.Memory/FileMappedResizableDirectMemory.cs, 4
- ./csharp/Platform.Memory/HeapResizableDirectMemory.cs, 6
- ./csharp/Platform.Memory/IArrayMemory.cs, 7
- ./csharp/Platform.Memory/IDirectMemory.cs, 7
- ./csharp/Platform.Memory/IMemory.cs, 8
- ./csharp/Platform.Memory/IResizableDirectMemory.cs, 8
- ./csharp/Platform.Memory/ResizableDirectMemoryBase.cs, 9
- ./csharp/Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs, 11