

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
15         /// <summary>
16         /// <para>
17         /// The array.
18         /// </para>
19         /// </summary>
20         private readonly TElement[] _array;
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"]/*' />
29         public long Size
30         {
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             get => _array.Length;
33         }
34
35         /// <inheritdoc/>
36         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem_
37         ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
38         public TElement this[long index]
39         {
40             [MethodImpl(MethodImplOptions.AggressiveInlining)]
41             get => _array[index];
42             [MethodImpl(MethodImplOptions.AggressiveInlining)]
43             set => _array[index] = value;
44         }
45
46         #endregion
47
48         #region Constructors
49
50         /// <summary>
51         /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}" />
52         ↪ class.</para>
53         /// <para>Инициализирует новый экземпляр класса <see
54         ↪ cref="ArrayMemory{TElement}" />.</para>
55         /// </summary>
56         /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         public ArrayMemory(long size) => _array = new TElement[size];
59
60         #endregion
61     }
62 }
```

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
    ↪ элемента.</para></typeparam>
14  public class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
    ↪ IArrayMemory<TElement>, IDirectMemory
15  where TElement : struct
16  {
17      #region Fields
18
19      /// <summary>
20      /// <para>
21      /// The memory.
22      /// </para>
23      /// <para></para>
24      /// </summary>
25      private readonly IDirectMemory _memory;
26
27      #endregion
28
29      #region Properties
30
31      /// <inheritdoc>
32      /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
    ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
33      public long Size
34      {
35          [MethodImpl(MethodImplOptions.AggressiveInlining)]
36          get => _memory.Size;
37      }
38
39      /// <inheritdoc>
40      /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
    ↪ path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*' />
41      public IntPtr Pointer
42      {
43          [MethodImpl(MethodImplOptions.AggressiveInlining)]
44          get => _memory.Pointer;
45      }
46
47      /// <inheritdoc>
48      /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
    ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
49      public TElement this[long index]
50      {
51          [MethodImpl(MethodImplOptions.AggressiveInlining)]
52          get => Pointer.ReadElementValue<TElement>(index);
53          [MethodImpl(MethodImplOptions.AggressiveInlining)]
54          set => Pointer.WriteElementValue(index, value);
55      }
56
57      #endregion
58
59      #region DisposableBase Properties
60
61      /// <inheritdoc>
62      protected override string ObjectName
63      {
64          [MethodImpl(MethodImplOptions.AggressiveInlining)]
65          get => $"Array as memory block at '{Pointer}' address.";
66      }
67
68      #endregion
69
70      #region Constructors
71
72      /// <summary>
73      /// <para>Initializes a new instance of the <see
    ↪ cref="DirectMemoryAsArrayMemoryAdapter{TElement}" /> class.</para>
74      /// <para>Инициализирует новый экземпляр класса <see
    ↪ cref="DirectMemoryAsArrayMemoryAdapter{TElement}" />.</para>
75      /// </summary>
76      /// <param name="memory"><para>An object implementing <see cref="IDirectMemory" />
    ↪ interface.</para><para>Объект, реализующий интерфейс <see
    ↪ 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)
    ↪ == 0, nameof(memory), "Memory is not aligned to element size.");

```

```

82     _memory = memory;
83 }
84
85 #endregion
86
87 #region DisposableBase Methods
88
89 /// <inheritdoc>
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 protected override void Dispose(bool manual, bool wasDisposed)
92 {
93     if (!wasDisposed)
94     {
95         _memory.DisposeIfPossible();
96     }
97 }
98
99 #endregion
100 }
101 }

```

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 индекатор 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
22         /// <summary>
23         /// <para>
24         /// The file.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         private readonly FileStream _file;
29
30         #endregion
31
32         #region Properties
33
34         /// <inheritdoc>
35         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
36         ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*' />
37         public long Size
38         {
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             get => _file.Length;
41         }
42
43         /// <inheritdoc>
44         /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
45         ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*' />
46         public TElement this[long index]
47         {
48             [MethodImpl(MethodImplOptions.AggressiveInlining)]
49             get
50             {
51                 _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
52                 return _file.ReadOrDefault<TElement>();
53             }
54             [MethodImpl(MethodImplOptions.AggressiveInlining)]
55             set
56             {
57                 _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
58                 _file.Write(value);
59             }
60         }
61     }
62 }

```

```

54     }
55 }
56
57 #endregion
58
59 #region DisposableBase Properties
60
61 /// <inheritdoc/>
62 protected override string ObjectName
63 {
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     get => $"File stored memory block at '{_file.Name}' path.";
66 }
67
68 #endregion
69
70 #region Constructors
71
72 /// <summary>
73 /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}" />
74   ↳ class.</para>
75 /// <para>Инициализирует новый экземпляр класса <see
76   ↳ cref="FileArrayMemory{TElement}" />.</para>
77 /// </summary>
78 /// <param name="file"><para>File stream.</para><para>Файловый поток.</para></param>
79 [MethodImpl(MethodImplOptions.AggressiveInlining)]
80 public FileArrayMemory(FileStream file) => _file = file;
81
82 /// <summary>
83 /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}" />
84   ↳ class.</para>
85 /// <para>Инициализирует новый экземпляр класса <see
86   ↳ cref="FileArrayMemory{TElement}" />.</para>
87 /// </summary>
88 /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
89 [MethodImpl(MethodImplOptions.AggressiveInlining)]
90 public FileArrayMemory(string path) : this(File.Open(path, FileMode.OpenOrCreate)) { }
91
92 #endregion
93
94 #region DisposableBase Methods
95
96 /// <inheritdoc/>
97 [MethodImpl(MethodImplOptions.AggressiveInlining)]
98 protected override void Dispose(bool manual, bool wasDisposed)
99 {
100     if (!wasDisposed)
101     {
102         _file.DisposeIfPossible();
103     }
104 }
105
106 #endregion
107 }
108 }

```

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
20         /// <summary>
21         /// <para>
22         /// The file.
23         /// </para>

```

```

24     /// <para></para>
25     /// </summary>
26     private MemoryMappedFile _file;
27     /// <summary>
28     /// <para>
29     /// The accessor.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     private MemoryMappedViewAccessor _accessor;
34
35     /// <summary>
36     /// <para>Gets path to memory mapped file.</para>
37     /// <para>Получает путь к отображенному в памяти файлу.</para>
38     /// </summary>
39     protected readonly string Path;
40
41     #endregion
42
43     #region DisposableBase Properties
44
45     /// <inheritdoc>
46     protected override string ObjectName
47     {
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         get => $"File stored memory block at '{Path}' path.";
50     }
51
52     #endregion
53
54     #region Constructors
55
56     /// <summary>
57     /// <para>Initializes a new instance of the <see
58     ↪ cref="FileMappedResizableDirectMemory"/> class.</para>
59     /// <para>Инициализирует новый экземпляр класса <see
60     ↪ cref="FileMappedResizableDirectMemory"/>.</para>
61     /// </summary>
62     /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
63     /// <param name="minimumReservedCapacity"><para>Minimum file size in
64     ↪ bytes.</para><para>Минимальный размер файла в байтах.</para></param>
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     public FileMappedResizableDirectMemory(string path, long minimumReservedCapacity)
67     {
68         Ensure.Always.ArgumentNotEmptyAndNotWhiteSpace(path, nameof(path));
69         if (minimumReservedCapacity < MinimumCapacity)
70         {
71             minimumReservedCapacity = MinimumCapacity;
72         }
73         Path = path;
74         var size = FileHelpers.GetSize(path);
75         ReservedCapacity = size > minimumReservedCapacity ? ((size /
76     ↪ minimumReservedCapacity) + 1) * minimumReservedCapacity :
77     ↪ minimumReservedCapacity;
78         UsedCapacity = size;
79     }
80
81     /// <summary>
82     /// <para>Initializes a new instance of the <see
83     ↪ cref="FileMappedResizableDirectMemory"/> class.</para>
84     /// <para>Инициализирует новый экземпляр класса <see
85     ↪ cref="FileMappedResizableDirectMemory"/>.</para>
86     /// </summary>
87     /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public FileMappedResizableDirectMemory(string path) : this(path, MinimumCapacity) { }
90
91     #endregion
92
93     #region Methods
94
95     /// <summary>
96     /// <para>
97     /// Maps the file using the specified capacity.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="capacity">
102    /// <para>The capacity.</para>

```

```

96     /// <para></para>
97     /// </param>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     private void MapFile(long capacity)
100     {
101         if (Pointer != IntPtr.Zero)
102         {
103             return;
104         }
105         _file = MemoryMappedFile.CreateFromFile(Path, FileMode.OpenOrCreate, mapName: null,
106             ↪ capacity, MemoryMappedFileAccess.ReadWrite);
107         _accessor = _file.CreateViewAccessor();
108         byte* pointer = null;
109         _accessor.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
110         Pointer = new IntPtr(pointer);
111     }
112     /// <summary>
113     /// <para>
114     /// Unmaps the file.
115     /// </para>
116     /// <para></para>
117     /// </summary>
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]
119     private void UnmapFile()
120     {
121         if (UnmapFile(Pointer))
122         {
123             Pointer = IntPtr.Zero;
124         }
125     }
126     /// <summary>
127     /// <para>
128     /// Determines whether this instance unmap file.
129     /// </para>
130     /// <para></para>
131     /// <para></para>
132     /// </summary>
133     /// <param name="pointer">
134     /// <para>The pointer.</para>
135     /// <para></para>
136     /// </param>
137     /// <returns>
138     /// <para>The bool</para>
139     /// <para></para>
140     /// </returns>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     private bool UnmapFile(IntPtr pointer)
143     {
144         if (pointer == IntPtr.Zero)
145         {
146             return false;
147         }
148         if (_accessor != null)
149         {
150             _accessor.SafeMemoryMappedViewHandle.ReleasePointer();
151             Disposable.TryDisposeAndResetToDefault(ref _accessor);
152         }
153         Disposable.TryDisposeAndResetToDefault(ref _file);
154         return true;
155     }
156     #endregion
157     #region ResizableDirectMemoryBase Methods
158     /// <inheritdoc>
159     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
160     ↪ path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv
161     ↪ edCapacityChanged(System.Int64,System.Int64)"]/*' />
162     [MethodImpl(MethodImplOptions.AggressiveInlining)]
163     protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
164     ↪ newReservedCapacity)
165     {
166         UnmapFile();
167         FileHelpers.SetSize(Path, newReservedCapacity);
168         MapFile(newReservedCapacity);
169     }

```

```

170
171     /// <inheritdoc/>
172     /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
    → path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP
    → ointer(System.IntPtr,System.Int64)"]/*' />
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void DisposePointer(IntPtr pointer, long usedCapacity)
175     {
176         if (UnmapFile(pointer))
177         {
178             FileHelpers.SetSize(Path, usedCapacity);
179         }
180     }
181
182     #endregion
183 }
184 }

```

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>
49        /// <para>Инициализирует новый экземпляр класса <see
50        → cref="HeapResizableDirectMemory"/>.</para>
51        /// </summary>
52        [MethodImpl(MethodImplOptions.AggressiveInlining)]
53        public HeapResizableDirectMemory() : this(MinimumCapacity) { }
54
55        #endregion
56
57        #region ResizableDirectMemoryBase Methods
58
59        /// <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 индекатор.</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
7     ///   ↳ pointers).</para>
8     /// <para>Представляет интерфейс блока памяти с изменяемым размером и прямым доступом (через
9     ///   ↳ неуправляемые указатели).</para>
10    /// </summary>
11    public interface IResizableDirectMemory : IDirectMemory
12    {
13        /// <summary>
14        /// <para>Gets or sets the reserved capacity in bytes of this memory block.</para>
15        /// <para>Возвращает или устанавливает зарезервированный размер блока памяти в
16        ///   ↳ байтах.</para>
17        /// </summary>
18        /// <remarks>
19        /// <para>
20        ///   If less then zero the value is replaced with zero.
21        ///   Cannot be less than the used capacity of this memory block.
22        /// </para>
23        /// <para>
24        ///   Если меньше нуля, значение заменяется на ноль.
25        ///   Не может быть меньше используемой емкости блока памяти.
26        /// </para>
27        /// </remarks>
28        long ReservedCapacity
29        {
30            [MethodImpl(MethodImplOptions.AggressiveInlining)]
31            get;
32            [MethodImpl(MethodImplOptions.AggressiveInlining)]
33            set;
34        }
35
36        /// <summary>
37        /// <para>Gets or sets the used capacity in bytes of this memory block.</para>
38        /// <para>Возвращает или устанавливает используемый размер в блоке памяти (в
39        ///   ↳ байтах).</para>
40        /// </summary>
41        /// <remarks>
42        /// <para>
43        ///   If less then zero the value is replaced with zero.
44        ///   Cannot be greater than the reserved capacity of this memory block.
45        /// </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
28         /// <summary>
29         /// <para>
30         /// The pointer.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private IntPtr _pointer;
35         /// <summary>
36         /// <para>
37         /// The reserved capacity.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         private long _reservedCapacity;
42         /// <summary>
43         /// <para>
44         /// The used capacity.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         private long _usedCapacity;
49
50         #endregion
51
52         #region Properties
53

```

```

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

```

```

116 {
117     [MethodImpl(MethodImplOptions.AggressiveInlining)]
118     get
119     {
120         Ensure.Always.NotDisposed(this);
121         return _usedCapacity;
122     }
123     [MethodImpl(MethodImplOptions.AggressiveInlining)]
124     set
125     {
126         Ensure.Always.NotDisposed(this);
127         if (value != _usedCapacity)
128         {
129             Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
130             _usedCapacity = value;
131         }
132     }
133 }
134
135 #endregion
136
137 #region DisposableBase Properties
138
139 /// <inheritdoc>
140 protected override bool AllowMultipleDisposeCalls
141 {
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     get => true;
144 }
145
146 #endregion
147
148 #region Methods
149
150 /// <summary>
151 /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
152   ↳ property.</para>
153 /// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
154 /// </summary>
155 /// <param name="oldReservedCapacity"><para>The old reserved capacity of the memory
156   ↳ block in bytes.</para><para>Старая зарезервированная емкость блока памяти в
157   ↳ байтах.</para></param>
158 /// <param name="newReservedCapacity"><para>The new reserved capacity of the memory
159   ↳ block in bytes.</para><para>Новая зарезервированная емкость блока памяти в
160   ↳ байтах.</para></param>
161 [MethodImpl(MethodImplOptions.AggressiveInlining)]
162 protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
163   ↳ newReservedCapacity);
164
165 /// <summary>
166 /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
167 /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
168 /// </summary>
169 /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
170   ↳ блок памяти.</para></param>
171 /// <param name="usedCapacity"><para>The used capacity of the memory block in
172   ↳ bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
173 [MethodImpl(MethodImplOptions.AggressiveInlining)]
174 protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
175
176 #endregion
177
178 #region DisposableBase Methods
179
180 /// <inheritdoc>
181 [MethodImpl(MethodImplOptions.AggressiveInlining)]
182 protected override void Dispose(bool manual, bool wasDisposed)
183 {
184     if (!wasDisposed)
185     {
186         var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
187         if (pointer != IntPtr.Zero)
188         {
189             DisposePointer(pointer, _usedCapacity);
190         }
191     }
192 }
193
194 #endregion

```

```
187     }
188 }
```

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
30     /// <summary>
31     /// <para>
32     /// Gets the last byte using the specified heap memory.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     /// <param name="heapMemory">
37     /// <para>The heap memory.</para>
38     /// <para></para>
39     /// </param>
40     /// <returns>
41     /// <para>The byte</para>
42     /// <para></para>
43     /// </returns>
44     private static byte GetLastByte(HeapResizableDirectMemory heapMemory)
45     {
46         var pointer1 = (void*)heapMemory.Pointer;
47         return *((byte*)pointer1 + heapMemory.ReservedCapacity - 1);
48     }
49 }
50 }

```

Index

- ./csharp/Platform.Memory.Tests/HeapResizableDirectMemoryTests.cs, 13
- ./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, 7
- ./csharp/Platform.Memory/IArrayMemory.cs, 8
- ./csharp/Platform.Memory/IDirectMemory.cs, 8
- ./csharp/Platform.Memory/IMemory.cs, 9
- ./csharp/Platform.Memory/IResizableDirectMemory.cs, 9
- ./csharp/Platform.Memory/ResizableDirectMemoryBase.cs, 10
- ./csharp/Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs, 13