```
LinksPlatform's Platform Memory Class Library
     ./csharp/Platform.Memory/ArrayMemory.cs
   using System.Runtime.CompilerServices;
   namespace Platform. Memory
4
   {
       /// <summary>
5
       /// <para>Represents a memory block with access via indexer.</para>
       /// <para>Представляет блок памяти с доступом через индексатор.</para>
       /// </summary>
       /// <typeparam name="TElement"><para>Element type.</para><para>Тип

→ элемента.</para></typeparam>

       public class ArrayMemory<TElement> : IArrayMemory<TElement>
10
11
12
            #region Fields
           private readonly TElement[] _array;
13
            #endregion
15
16
            #region Properties
17
18
            /// <inheritdoc/>
19
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
20
                path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
           public long Size
21
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
                get => _array.Length;
24
            }
25
26
            /// <inheritdoc/>
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem |
28
               ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
           public TElement this[long index]
29
30
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
                get => _array[index];
32
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
33
                set => _array[index] = value;
35
36
            #endregion
37
            #region Constuctors
39
40
            /// <summary>
41
            /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}"/>
               class.</para>
            /// <para>Инициализирует новый экземпляр класса <see
            /// </summary>
44
            /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
45
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
46
           public ArrayMemory(long size) => _array = new TElement[size];
47
48
            #endregion
49
       }
50
   }
51
1.2
    ./csharp/Platform.Memory/DirectMemoryAsArrayMemoryAdapter.cs
   using System;
   using System.Runtime.CompilerServices;
   using Platform.Disposables;
using Platform.Exceptions;
using Platform.Unsafe;
4
   namespace Platform. Memory
8
        /// <summary>
9
       /// <para>Represents adapter to a memory block with access via indexer.</para>
10
       /// <para>Представляет адаптер к блоку памяти с доступом через индексатор.</para>
11
       /// </summary>
       /// <typeparam name="TElement"><para>Element type.</para><para>Тип
13
           элемента.</para></typeparam>
       public class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
14
           IArrayMemory<TElement>, IDirectMemory
15
           where TElement : struct
16
            #region Fields
17
```

```
private readonly IDirectMemory _memory;
18
19
            #endregion
21
            #region Properties
22
            /// <inheritdoc/>
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
25
               path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
            public long Size
26
27
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
28
                get => _memory.Size;
30
            /// <inheritdoc/>
32
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
33
               path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
            public IntPtr Pointer
35
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
36
37
                get => _memory.Pointer;
            }
38
39
            /// <inheritdoc/>
40
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
41
               ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
            public TElement this[long index]
42
43
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
44
                get => Pointer.ReadElementValue<TElement>(index);
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
                set => Pointer.WriteElementValue(index, value);
47
48
49
            #endregion
51
            #region DisposableBase Properties
53
            /// <inheritdoc/>
            protected override string ObjectName
5.5
56
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
57
                get => $ "Array as memory block at '{Pointer}' address.";
58
59
            #endregion
61
62
            #region Constructors
64
            /// <summary>
65
            /// <para>Initializes a new instance of the <see
66
                cref="DirectMemoryAsArrayMemoryAdapter{TElement}"/> class.
            /// <para>Инициализирует новый экземпляр класса <see
                cref="DirectMemoryAsArrayMemoryAdapter{TElement}"/>.</para>
            /// </summary>
68
            /// <param name="memory"><para>An object implementing <see cref="IDirectMemory"/>
            \hookrightarrow interface.</para><para>Объект, реализующий интерфейс <see
               cref="IDirectMemory"/>.</para></param>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
70
            public DirectMemoryAsArrayMemoryAdapter(IDirectMemory memory)
71
                Ensure.Always.ArgumentNotNull(memory, nameof(memory));
73
                Ensure.Always.ArgumentMeetsCriteria(memory, m => (m.Size % Structure<TElement>.Size)
74

⇒ == 0, nameof(memory), "Memory is not aligned to element size.");
                _memory = memory;
            }
76
77
            #endregion
78
79
            #region DisposableBase Methods
80
81
            /// <inheritdoc/>
82
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected override void Dispose(bool manual, bool wasDisposed)
84
85
                if (!wasDisposed)
86
                    _memory.DisposeIfPossible();
88
```

```
89
91
            #endregion
92
       }
93
   }
94
    ./csharp/Platform.Memory/FileArrayMemory.cs
1.3
   using System.IO;
   using System.Runtime.CompilerServices;
   using Platform.Disposables;
   using Platform.Unsafe;
4
   using Platform.IO;
   namespace Platform. Memory
8
        /// <summary>
       /// <para>Represents a memory block with access via indexer and stored as file on
10
           disk.</para>
       /// <para>\Pi ar{p}едставляет блок памяти с доступом через индексатор и хранящийся в виде файла на
11
       12
       /// <typeparam name="TElement"><para>Element type.</para><para>Тип
13
           элемента.</para></typeparam>
       public class FileArrayMemory<TElement> : DisposableBase, IArrayMemory<TElement> //-V3073
            where TElement : struct
15
16
            #region Fields
17
            private readonly FileStream _file;
19
            #endregion
20
            #region Properties
22
23
            /// <inheritdoc/>
24
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
25
               path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
            public long Size
26
27
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
28
                get => _file.Length;
29
            }
30
31
            /// <inheritdoc/>
32
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
33
                ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
            public TElement this[long index]
35
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
36
37
                get
38
                    _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                    return _file.ReadOrDefault<TElement>();
40
41
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
42
                set
43
                {
44
                    _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                    _file.Write(value);
46
47
            }
49
            #endregion
50
51
            #region DisposableBase Properties
52
53
            /// <inheritdoc/>
54
            protected override string ObjectName
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
57
                get => $|"File stored memory block at '{_file.Name}' path.";
58
59
60
            #endregion
62
            #region Contructors
63
64
            /// <summary>
            /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}"/>

→ class.</para>
```

```
/// <para>Инициализирует новый экземпляр класса <see
                cref="FileArrayMemory{TElement}"/>.</para>
            /// </summary>
            /// <param name="file"><para>File stream.</para><para>Файловый поток.</para></param>
69
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
7.0
            public FileArrayMemory(FileStream file) => _file = file;
71
72
            /// <summary>
73
            /// <para>Initializes a new instance of the <see cref="FileArrayMemory{TElement}"/>
               class.</para>
            /// <para>Инициализирует новый экземпляр класса <see
                cref="FileArrayMemory{TElement}"/>.</para>
            /// </summary>
76
            /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
77
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public FileArrayMemory(string path) : this(File.Open(path, FileMode.OpenOrCreate)) { }
79
80
            #endregion
81
            #region DisposableBase Methods
83
84
            /// <inheritdoc/>
8.5
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected override void Dispose(bool manual, bool wasDisposed)
87
88
                if (!wasDisposed)
89
                {
                     _file.DisposeIfPossible();
91
92
            }
94
            #endregion
95
        }
96
97
     ./csharp/Platform.Memory/FileMappedResizableDirectMemory.cs
1.4
   using System;
   using System.IO;
   using System.IO.MemoryMappedFiles;
3
   using System.Runtime.CompilerServices;
   using Platform.Disposables;
   using Platform.Exceptions; using Platform.Collections;
   using Platform. IO;
   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
16
        public unsafe class FileMappedResizableDirectMemory : ResizableDirectMemoryBase
17
            #region Fields
18
            private MemoryMappedFile _file;
19
            private MemoryMappedViewAccessor _accessor;
21
            /// <summary>
            /// <para>Gets path to memory mapped file.</para>
23
            /// <para>Получает путь к отображенному в памяти файлу.</para>
24
            /// </summary>
25
            protected readonly string Path;
26
            #endregion
28
29
            #region DisposableBase Properties
30
31
32
            /// <inheritdoc/>
            protected override string ObjectName
33
34
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
35
                get => |$|"File stored memory block at '{Path}' path.";
37
38
            #endregion
39
            #region Constructors
41
42
            /// <summary>
43
            /// <para>Initializes a new instance of the <see
               cref="FileMappedResizableDirectMemory"/> class.</para>
```

```
/// <para>Инициализирует новый экземпляр класса <see
45
                 cref="FileMappedResizableDirectMemory"/>.</para>
             /// </summary>
46
             /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
47
             /// <param name="minimumReservedCapacity"><para>Minimum file size in
48
                bytes.</para><para>Минимальный размер файла в байтах.</para></param>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
49
            public FileMappedResizableDirectMemory(string path, long minimumReservedCapacity)
51
                 Ensure.Always.ArgumentNotEmptyAndNotWhiteSpace(path, nameof(path));
52
                 if (minimumReservedCapacity < MinimumCapacity)</pre>
53
                 {
54
                     minimumReservedCapacity = MinimumCapacity;
55
                 Path = path;
57
58
                 var size = FileHelpers.GetSize(path);
                 ReservedCapacity = size > minimumReservedCapacity ? ((size /
59

→ minimumReservedCapacity) + 1) * minimumReservedCapacity :

                     minimumReservedCapacity;
                 UsedCapacity = size;
60
             }
61
62
             /// <summary>
63
             /// <para>Initializes a new instance of the <see
                cref="FileMappedResizableDirectMemory"/> class.</para>
             /// <para>Инициализирует новый экземпляр класса <see
                cref="FileMappedResizableDirectMemory"/>.</para>
             /// </summary>
66
             /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
67
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public FileMappedResizableDirectMemory(string path) : this(path, MinimumCapacity) { }
69
70
             #endregion
7.1
72
             #region Methods
73
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
             private void MapFile(long capacity)
7.5
76
                 if (Pointer != IntPtr.Zero)
77
                 {
                     return;
79
                 }
                 _file = MemoryMappedFile.CreateFromFile(Path, FileMode.OpenOrCreate, mapName: null,
81
                 \  \  \, \rightarrow \  \  \, \text{capacity, MemoryMappedFileAccess.ReadWrite)};
                  _accessor = _file.CreateViewAccessor();
82
                 byte* pointer = null;
83
                 _accessor.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
84
                 Pointer = new IntPtr(pointer);
85
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
87
            private void UnmapFile()
88
89
                 if (UnmapFile(Pointer))
90
                 {
91
                     Pointer = IntPtr.Zero;
93
94
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
             private bool UnmapFile(IntPtr pointer)
96
97
                 if (pointer == IntPtr.Zero)
98
                 {
                     return false;
100
                 }
                 if (_accessor != null)
102
                 {
103
                      _accessor.SafeMemoryMappedViewHandle.ReleasePointer();
104
                     Disposable.TryDisposeAndResetToDefault(ref _accessor);
106
                 Disposable.TryDisposeAndResetToDefault(ref _file);
107
108
                 return true;
109
110
             #endregion
111
112
             #region ResizableDirectMemoryBase Methods
114
             /// <inheritdoc/>
```

```
/// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
116
             _{\rightarrow} \quad \texttt{path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv}_{\bot}
                 edCapacityChanged(System.Int64,System.Int64)"]/*'/>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
117
            protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
                 newReservedCapacity)
119
                 UnmapFile();
120
                 FileHelpers.SetSize(Path, newReservedCapacity);
                 MapFile(newReservedCapacity);
123
124
             /// <inheritdoc/>
125
             /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
126
                path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.DisposeP_
                ointer(System.IntPtr,System.Int64)"]/*'/>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected override void DisposePointer(IntPtr pointer, long usedCapacity)
129
                 if (UnmapFile(pointer))
130
                 {
                     FileHelpers.SetSize(Path, usedCapacity);
132
133
             }
134
135
             #endregion
136
        }
137
138
      ./csharp/Platform.Memory/HeapResizableDirectMemory.cs
1.5
    using System;
    using System.Runtime.CompilerServices;
          System.Runtime.InteropServices;
 3
    using
    using Platform.Unsafe;
    namespace Platform. Memory
 6
 7
         /// <summary>
        /// <para>Represents a memory block allocated in Heap.</para>
 9
        /// <para>Представляет блок памяти, выделенный в "куче".</para>
10
        /// </summary>
11
        public unsafe class HeapResizableDirectMemory : ResizableDirectMemoryBase
12
13
             #region DisposableBase Properties
14
             /// <inheritdoc/>
16
            protected override string ObjectName
17
18
                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
19
                 get => $\frac{\$}{\}\]"Heap stored memory block at {Pointer} address.";
20
22
23
             #endregion
24
             #region Constructors
25
             /// <summary>
27
             /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
2.8
                class.</para>
             /// <para>Инициализирует новый экземпляр класса <see
2.9
                cref="HeapResizableDirectMemory"/>.</para>
             /// </summary>
30
             /// <param name="minimumReservedCapacity"><para>Minimum file size in
31
                bytes.</para><para>Минимальный размер файла в байтах.</para></param>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32
            public HeapResizableDirectMemory(long minimumReservedCapacity)
33
                 if (minimumReservedCapacity < MinimumCapacity)</pre>
35
                 {
36
                     minimumReservedCapacity = MinimumCapacity;
37
38
                 ReservedCapacity = minimumReservedCapacity;
39
                 UsedCapacity = 0;
40
             }
42
             /// <summary>
43
             /// <para>Initializes a new instance of the <see cref="HeapResizableDirectMemory"/>
44
                class.</para>
```

```
/// <para>Инициализирует новый экземпляр класса <see
45
               cref="HeapResizableDirectMemory"/>.</para>
            /// </summary>
46
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
47
           public HeapResizableDirectMemory() : this(MinimumCapacity) { }
48
            #endregion
50
51
            #region ResizableDirectMemoryBase Methods
53
            /// <inheritdoc/>
54
            //// <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)]
            protected override void DisposePointer(IntPtr pointer, long usedCapacity) =>

→ Marshal.FreeHGlobal(pointer);
            /// <inheritdoc/>
59
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
60
              path='doc/members/member[@name="M:Platform.Memory.ResizableDirectMemoryBase.OnReserv_
               edCapacityChanged(System.Int64,System.Int64)"]/*'/>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
           protected override void OnReservedCapacityChanged(long oldReservedCapacity, long
               newReservedCapacity)
63
                if (Pointer == IntPtr.Zero)
64
                    Pointer = Marshal.AllocHGlobal(new IntPtr(newReservedCapacity));
                    MemoryBlock.Zero((void*)Pointer, newReservedCapacity);
67
                }
68
                else
69
                {
70
                    Pointer = Marshal.ReAllocHGlobal(Pointer, new IntPtr(newReservedCapacity));
                    var pointer = (byte*)Pointer + oldReservedCapacity;
72
                    MemoryBlock.Zero(pointer, newReservedCapacity - oldReservedCapacity);
73
            }
75
76
            #endregion
77
       }
78
79
    ./csharp/Platform.Memory/IArrayMemory.cs
   using System.Runtime.CompilerServices;
1
2
   namespace Platform. Memory
4
   {
        /// <summary>
5
        /// <para>Represents a memory block interface with access via indexer.</para>
        /// <para>Представляет интерфейс блока памяти с доступом через индексатор.</para>
       /// </summary>
       /// <typeparam name="TElement"><para>Element type.</para><para>Тип
           элемента.</para></typeparam>
       public interface IArrayMemory<TElement> : IMemory
10
1.1
            /// <summary>
12
            /// <para>Gets or sets the element at the specified index.</para>
            /// <para>Возвращает или устанавливает элемент по указанному индексу.</para>
14
            /// </summary>
15
            /// <param name="index"><para>The index of the element to get or set.</para><para>Индекс
16
            → элемента, который нужно получить или установить.</para></param>
            TElement this [long index]
18
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
19
20
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
2.1
                set;
            }
23
       }
24
     ./csharp/Platform.Memory/IDirectMemory.cs
1.7
   using System;
   using System.Runtime.CompilerServices;
2
   namespace Platform. Memory
4
```

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

```
/// <summary>
            /// <para>Gets or sets the used capacity in bytes of this memory block.</para>
34
            /// <para>Возвращает или устанавливает используемый размер в блоке памяти (в
35
               байтах).</para>
            /// </summary>
36
            /// <remarks>
            /// <para>
/// If less then zero the value is replaced with zero.
38
39
            /// Cannot be greater than the reserved capacity of this memory block.
40
            /// </para>
41
            /// <para>
42
            /// It is recommended to reduce the reserved capacity of the memory block to the used
43
            → capacity (specified in this property) after the completion of the use of the memory
               block.
            /// </para>
44
            /// <para>
45
            /// \bar{\text{Ec}}ли меньше нуля, значение заменяется на ноль.
46
            /// Не может быть больше, чем зарезервированная емкость этого блока памяти.
47
            /// </para>
48
            /// <para>
            /// Рекомендуется уменьшать фактический размер блока памяти до используемого размера
50
               (указанного в этом свойстве) после завершения использования блока памяти.
            /// </para>
51
            /// </remarks>
52
            long UsedCapacity
53
54
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
56
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
                set;
58
            }
       }
60
   }
61
1.10 ./csharp/Platform.Memory/ResizableDirectMemoryBase.cs
   using System;
   using System. Threading
   using System.Runtime.CompilerServices;
   using Platform.Exceptions;
   using Platform.Disposables;
5
   using Platform.Ranges;
   namespace Platform.Memory
9
        /// <summary>
10
        /// <para>Provides a base implementation for the resizable memory block with direct access
11
            (via unmanaged pointers)./para>
        /// <para>Предоставляет базовую реализацию для блока памяти с изменяемым размером и прямым
12
            доступом (через неуправляемые указатели).</para>
        /// </summary>
13
       public abstract class ResizableDirectMemoryBase : DisposableBase, IResizableDirectMemory
15
            #region Constants
17
            /// <summary>
            /// <para>Gets minimum capacity in bytes.</para>
19
            /// <para>Возвращает минимальную емкость в байтах.</para>
20
            /// </summary>
21
            public static readonly long MinimumCapacity = Environment.SystemPageSize;
22
            #endregion
24
25
            #region Fields
            private IntPtr _pointer;
27
            private long _reservedCapacity;
private long _usedCapacity;
28
29
30
            #endregion
32
            #region Properties
34
            /// <inheritdoc/>
            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
36
                path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
            /// <exception cref="ObjectDisposedException"><para>The memory block is
37
                disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
            public long Size
39
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
Ensure.Always.NotDisposed(this);
        return UsedCapacity;
    }
/// <inheritdoc/>
/// <include file='bin\Release\netstandard2.0\Platform.Memory.xml
   path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
/// <exception cref="ObjectDisposedException"><para>The memory block is
🛶 disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
public IntPtr Pointer
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
        Ensure.Always.NotDisposed(this);
       return _pointer;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected set
        Ensure.Always.NotDisposed(this);
        _pointer = value;
    }
}
/// <inheritdoc/>
///<include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
   ber[@name="P:Platform.Memory.IResizableDirectMemory.ReservedCapacity"]/*'/>
/// <exception cref="ObjectDisposedException"><para>The memory block is
   disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
/// <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>
public long ReservedCapacity
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get
        Ensure.Always.NotDisposed(this);
        return _reservedCapacity;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    set
    {
        Ensure.Always.NotDisposed(this);
        if (value != _reservedCapacity)
            Ensure.Always.ArgumentInRange(value, new Range<long>(_usedCapacity,
            → long.MaxValue));
            OnReservedCapacityChanged(_reservedCapacity, value);
            _reservedCapacity = value;
        }
    }
}
/// <inheritdoc/>
/// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
   ber[@name="P:Platform.Memory.IResizableDirectMemory.UsedCapacity"]/*'/>
/// <exception cref="ObjectDisposedException"><para>The memory block is
   disposed.</para>Блок памяти уже высвобожден.</para></exception>
/// <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>- Была выполнена попытка установить используемую емкость на
   значение, которое больше, чем зарезервированная емкость или меньше
   нуля.</para></exception>
public long UsedCapacity
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get
       Ensure.Always.NotDisposed(this);
        return _usedCapacity;
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

41

43

45 46 47

48

49

50

51 52

54

56

57 58

59

61

62 63

64

65 66

67

7.0

71

73

76

78

79

80

81

82 83

84

85

87

88

90

92

93

94

96

100

101

103

104

```
set
105
                     Ensure.Always.NotDisposed(this);
107
                     if (value != _usedCapacity)
108
109
                         Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
110
                          _usedCapacity = value;
111
                     }
112
                 }
113
             }
114
115
             #endregion
116
117
             #region DisposableBase Properties
119
             /// <inheritdoc/>
            protected override bool AllowMultipleDisposeCalls
121
122
                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
123
                 get => true;
124
             }
125
126
             #endregion
127
128
             #region Methods
129
130
             /// <summary>
131
             /// <para>Executed on the event of change for <see cref="ReservedCapacity"/>
132
             → property.</para>
/// <para>Выполняется в случае изменения свойства <see cref="ReservedCapacity"/>.</para>
             /// </summary>
134
             /// <param name="oldReservedCapacity"><para>The old reserved capacity of the memory
135
                block in bytes.</para><para>Старая зарезервированная емкость блока памяти в
                 байтах.</para></param>
             /// <param name="newReservedCapacity"><para>The new reserved capacity of the memory
136
                block in bytes.</para><para>Новая зарезервированная емкость блока памяти в
                байтах.</para></param>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
137
            protected abstract void OnReservedCapacityChanged(long oldReservedCapacity, long
138
             → newReservedCapacity);
             /// <summary>
140
             /// <para>Executed when it is time to dispose <see cref="Pointer"/>.</para>
141
             /// <para>Выполняется, когда пришло время высвободить <see cref="Pointer"/>.</para>
142
             /// </summary>
             /// <param name="pointer"><para>The pointer to a memory block.</para><para>Указатель на
144
                блок памяти.</para></param>
             /// <param name="usedCapacity"><para>The used capacity of the memory block in
145
                bytes.</para><para>Используемая емкость блока памяти в байтах.</para></param>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
146
            protected abstract void DisposePointer(IntPtr pointer, long usedCapacity);
147
148
             #endregion
149
             #region DisposableBase Methods
151
152
             /// <inheritdoc/>
153
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
154
             protected override void Dispose(bool manual, bool wasDisposed)
155
156
                 if (!wasDisposed)
157
158
                     var pointer = Interlocked.Exchange(ref _pointer, IntPtr.Zero);
159
                        (pointer != IntPtr.Zero)
160
161
                         DisposePointer(pointer, _usedCapacity);
                     }
163
                 }
164
             }
166
             #endregion
167
        }
168
169
      ./csharp/Platform.Memory/TemporaryFileMappedResizableDirectMemory.cs
1.11
    using System. IO:
    using System.Runtime.CompilerServices;
    namespace Platform. Memory
 4
```

```
5
        /// <summary>
       /// <para>Represents a memory block stored as a temporary file on disk.</para>
       /// <para>Представляет блок памяти, хранящийся в виде временного файла на диске.</para>
       /// </summary>
       public class TemporaryFileMappedResizableDirectMemory : FileMappedResizableDirectMemory
10
11
            #region DisposableBase Properties
12
            /// <inheritdoc/>
14
           protected override string ObjectName
15
16
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
17
                get => $\"Temporary file stored memory block at '{Path}' path.";
18
19
20
            #endregion
22
            #region Constructors
23
24
            /// <summary>
25
            /// <para>Initializes a new instance of the <see
26
               cref="TemporaryFileMappedResizableDirectMemory"/> class.
            /// <para>Инициализирует новый экземпляр класса <see
               cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
            /// </summary>
            /// <param name="minimumReservedCapacity"><para>Minimum file size in
            🛶 bytes.</para><para>Минимальный размер файла в байтах.</para></param>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
30
           public TemporaryFileMappedResizableDirectMemory(long minimumReservedCapacity) :
31
            ⇒ base(System.IO.Path.GetTempFileName(), minimumReservedCapacity) { }
            /// <summary>
33
            /// <para>Initializes a new instance of the <see
34
               cref="TemporaryFileMappedResizableDirectMemory"/> class.
            /// <para>Инициализирует новый экземпляр класса <see
               cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
            /// </summary>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
37
           public TemporaryFileMappedResizableDirectMemory() : this(MinimumCapacity) { }
38
39
            #endregion
41
            #region DisposableBase Methods
42
43
            /// <inheritdoc/>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
45
            protected override void Dispose(bool manual, bool wasDisposed)
46
47
                base.Dispose(manual, wasDisposed);
48
                if (!wasDisposed)
49
                    File.Delete(Path);
52
53
54
            #endregion
       }
56
57
     ./csharp/Platform. Memory. Tests/HeapResizable Direct Memory Tests.cs\\
1.12
   using Xunit;
1
2
   namespace Platform. Memory. Tests
       public unsafe class HeapResizableDirectMemoryTests
5
            |Fact|
           public void CorrectMemoryReallocationTest()
                using var heapMemory = new HeapResizableDirectMemory();
                var value1 = GetLastByte(heapMemory);
11
                heapMemory.ReservedCapacity *= 2;
                var value2 = GetLastByte(heapMemory);
13
                Assert.Equal(value1, value2);
14
                Assert.Equal(0, value1);
15
           private static byte GetLastByte(HeapResizableDirectMemory heapMemory)
17
```

```
var pointer1 = (void*)heapMemory.Pointer;
return *((byte*)pointer1 + heapMemory.ReservedCapacity - 1);
}
21     }
22    }
23 }
```

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