LinksPlatform's Platform.Memory Class Library

./ArrayMemory.cs

```csharp
namespace Platform.Memory
{
    /// <summary>
    /// <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>
    {
        #region Fields

        private readonly TElement[] _array;

        #endregion

        #region Properties

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
        public long Size => _array.Length;

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem↓
        ↪ ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
        public TElement this[long index]
        {
            get => _array[index];
            set => _array[index] = value;
        }

        #endregion

        #region Constuctors

        /// <summary>
        /// <para>Initializes a new instance of the <see cref="ArrayMemory{TElement}"/>
        ↪ class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪ cref="ArrayMemory{TElement}"/>.</para>
        /// </summary>
        /// <param name="size"><para>Size in bytes.</para><para>Размер в байтах.</para></param>
        public ArrayMemory(long size) => _array = new TElement[size];

        #endregion
    }
}
```

./DirectMemoryAsArrayMemoryAdapter.cs

```csharp
using System;
using Platform.Disposables;
using Platform.Exceptions;
using Platform.Unsafe;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents adapter to a memory block with access via indexer.</para>
    /// <para>Представляет адаптер к блоку памяти с доступом через индексатор.</para>
    /// </summary>
    /// <typeparam name="TElement"><para>Element type.</para><para>Тип
    ↪ элемента.</para></typeparam>
    public class DirectMemoryAsArrayMemoryAdapter<TElement> : DisposableBase,
    ↪ IArrayMemory<TElement>, IDirectMemory
        where TElement : struct
    {
        #region Fields

        private readonly IDirectMemory _memory;

        #endregion

        #region Properties

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪ path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
```

```csharp
        public long Size => _memory.Size;

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪  path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
        public IntPtr Pointer => _memory.Pointer;

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem⌋
        ↪  ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
        public TElement this[long index]
        {
            get => Pointer.GetElement(Structure<TElement>.Size, index).GetValue<TElement>();
            set => Pointer.GetElement(Structure<TElement>.Size, index).SetValue(value);
        }

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName => $"Array as memory block at '{Pointer}'
        ↪  address.";

        #endregion

        #region Constructors

        /// <summary>
        /// <para>Initializes a new instance of the <see
        ↪  cref="DirectMemoryAsArrayMemoryAdapter{TElement}"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪  cref="DirectMemoryAsArrayMemoryAdapter{TElement}"/>.</para>
        /// </summary>
        /// <param name="memory"><para>An object implementing <see cref="IDirectMemory"/>
        ↪  interface.</para><para>Объект, реализующий интерфейс <see
        ↪  cref="IDirectMemory"/>.</para></param>
        public DirectMemoryAsArrayMemoryAdapter(IDirectMemory memory)
        {
            Ensure.Always.ArgumentMeetsCriteria(m => (m.Size % Structure<TElement>.Size) == 0,
            ↪  memory, nameof(memory), "Memory is not aligned to element size.");
            _memory = memory;
        }

        #endregion

        #region DisposableBase Methods

        /// <inheritdoc/>
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                _memory.DisposeIfPossible();
            }
        }

        #endregion
    }
}
```

./FileArrayMemory.cs

```csharp
using System.IO;
using Platform.Disposables;
using Platform.Unsafe;
using Platform.IO;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block with access via indexer and stored as file on
    ↪  disk.</para>
    /// <para>Представляет блок памяти с доступом через индексатор и хранящийся в виде файла на
    ↪  диске.</para>
    /// </summary>
    /// <typeparam name="TElement"><para>Element type.</para><para>Тип
    ↪  элемента.</para></typeparam>
    public class FileArrayMemory<TElement> : DisposableBase, IArrayMemory<TElement> //-V3073
        where TElement : struct
```

```csharp
    {
        #region Fields

        private readonly string _address;
        private readonly FileStream _file;

        #endregion

        #region Properties

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
        ↪   path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
        public long Size => _file.Length;

        /// <inheritdoc/>
        /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml' path='doc/members/mem
        ↪   ber[@name="P:Platform.Memory.IArrayMemory`1.Item(System.Int64)"]/*'/>
        public TElement this[long index]
        {
            get
            {
                _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                return _file.ReadOrDefault<TElement>();
            }
            set
            {
                _file.Seek(Structure<TElement>.Size * index, SeekOrigin.Begin);
                _file.Write(value);
            }
        }

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName => $"File stored memory block at '{_address}'
        ↪   path.";

        #endregion

        #region Contructors

        /// <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="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
        public FileArrayMemory(string path)
        {
            _address = path;
            _file = File.Open(path, FileMode.OpenOrCreate, FileAccess.ReadWrite);
        }

        #endregion

        #region DisposableBase Methods

        /// <inheritdoc/>
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if(!wasDisposed)
            {
                _file.DisposeIfPossible();
            }
        }

        #endregion
    }
}
```

./FileMappedResizableDirectMemory.cs

```csharp
using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using Platform.Disposables;
using Platform.Exceptions;
```

```csharp
using Platform.Collections;
using Platform.IO;

namespace Platform.Memory
{
    /// <summary>
    /// <para>Represents a memory block stored as a file on disk.</para>
    /// <para>Представляет блок памяти, хранящийся в виде файла на диске.</para>
    /// </summary>
    public unsafe class FileMappedResizableDirectMemory : ResizableDirectMemoryBase
    {
        #region Fields

        private MemoryMappedFile _file;
        private MemoryMappedViewAccessor _accessor;

        /// <summary>
        /// <para>Gets path to memory mapped file.</para>
        /// <para>Получает путь к отображенному в памяти файлу.</para>
        /// </summary>
        protected readonly string Path;

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override string ObjectName => $"File stored memory block at '{Path}' path.";

        #endregion

        #region Constructors

        /// <summary>
        /// <para>Initializes a new instance of the <see
        ///   cref="FileMappedResizableDirectMemory"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ///   cref="FileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        /// <param name="path"><para>An path to file.</para><para>Путь к файлу.</para></param>
        /// <param name="minimumReservedCapacity"><para>Minimum file size in
        ///   bytes.</para><para>Минимальный размер файла в байтах.</para></param>
        public FileMappedResizableDirectMemory(string path, long minimumReservedCapacity)
        {
            Ensure.Always.ArgumentNotEmptyAndNotWhiteSpace(path, nameof(path));
            if (minimumReservedCapacity < MinimumCapacity)
            {
                minimumReservedCapacity = MinimumCapacity;
            }
            Path = path;
            var size = FileHelpers.GetSize(Path);
            ReservedCapacity = size > minimumReservedCapacity ? ((size /
                minimumReservedCapacity) + 1) * minimumReservedCapacity :
                minimumReservedCapacity;
            UsedCapacity = size;
        }

        /// <summary>
        /// <para>Initializes a new instance of the <see
        ///   cref="FileMappedResizableDirectMemory"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ///   cref="FileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        /// <param name="address"><para>An path to file.</para><para>Путь к файлу.</para></param>
        public FileMappedResizableDirectMemory(string address) : this(address, MinimumCapacity)
            { }

        #endregion

        #region Methods

        private void MapFile(long capacity)
        {
            if (Pointer != IntPtr.Zero)
            {
                return;
            }
            _file = MemoryMappedFile.CreateFromFile(Path, FileMode.Open, mapName: null,
                capacity, MemoryMappedFileAccess.ReadWrite);
```

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

./HeapResizableDirectMemory.cs

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

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

./IArrayMemory.cs
```csharp
1   namespace Platform.Memory
2   {
3       /// <summary>
4       /// <para>Represents a memory block interface with access via indexer.</para>
5       /// <para>Представляет интерфейс блока памяти с доступом через индексатор.</para>
6       /// </summary>
7       /// <typeparam name="TElement"><para>Element type.</para><para>Тип
        ↪  элемента.</para></typeparam>
8       public interface IArrayMemory<TElement> : IMemory
9       {
10          /// <summary>
11          /// <para>Gets or sets the element at the specified index.</para>
12          /// <para>Возвращает или устанавливает элемент по указанному индексу.</para>
```

```csharp
13          /// </summary>
14          /// <param name="index"><para>The index of the element to get or set.</para><para>Индекс
    ↪ элемента, который нужно получить или установить.</para></param>
15          TElement this[long index] { get; set; }
16      }
17  }
```

./IDirectMemory.cs

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

./IMemory.cs

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

./IResizableDirectMemory.cs

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

```csharp
28            /// </summary>
29            /// <remarks>
30            /// <para>
31            /// If less then zero the value is replaced with zero.
32            /// Cannot be greater than the reserved capacity of this memory block.
33            /// </para>
34            /// <para>
35            /// It is recommended to reduce the reserved capacity of the memory block to the used
   ↪  capacity (specified in this property) after the completion of the use of the memory
   ↪  block.
36            /// </para>
37            /// <para>
38            /// Если меньше нуля, значение заменяется на ноль.
39            /// Не может быть больше, чем зарезервированная емкость этого блока памяти.
40            /// </para>
41            /// <para>
42            /// Рекомендуется уменьшать фактический размер блока памяти до используемого размера
   ↪  (указанного в этом свойстве) после завершения использования блока памяти.
43            /// </para>
44            /// </remarks>
45            long UsedCapacity { get; set; }
46        }
47    }
```

./ResizableDirectMemoryBase.cs

```csharp
1    using System;
2    using System.Threading;
3    using Platform.Exceptions;
4    using Platform.Disposables;
5    using Platform.Ranges;
6
7    namespace Platform.Memory
8    {
9        /// <summary>
10        /// <para>Provides a base implementation for the resizable memory block with direct access
   ↪  (via unmanaged pointers).</para>
11        /// <para>Предоставляет базовую реализацию для блока памяти с изменяемым размером и прямым
   ↪  доступом (через неуправляемые указатели).</para>
12        /// </summary>
13        public abstract class ResizableDirectMemoryBase : DisposableBase, IResizableDirectMemory
14        {
15            #region Constants
16
17            /// <summary>
18            /// <para>Gets minimum capacity in bytes.</para>
19            /// <para>Возвращает минимальную емкость в байтах.</para>
20            /// </summary>
21            public static readonly long MinimumCapacity = 4096;
22
23            #endregion
24
25            #region Fields
26
27            private IntPtr _pointer;
28            private long _reservedCapacity;
29            private long _usedCapacity;
30
31            #endregion
32
33            #region Properties
34
35            /// <inheritdoc/>
36            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
   ↪  path='doc/members/member[@name="P:Platform.Memory.IMemory.Size"]/*'/>
37            /// <exception cref="ObjectDisposedException"><para>The memory block is
   ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
38            public long Size
39            {
40                get
41                {
42                    Ensure.Always.NotDisposed(this);
43                    return UsedCapacity;
44                }
45            }
46
47            /// <inheritdoc/>
48            /// <include file='bin\Release\netstandard2.0\Platform.Memory.xml'
   ↪  path='doc/members/member[@name="P:Platform.Memory.IDirectMemory.Pointer"]/*'/>
49            /// <exception cref="ObjectDisposedException"><para>The memory block is
   ↪  disposed.</para><para>Блок памяти уже высвобожден.</para></exception>
```

```csharp
        public IntPtr Pointer
        {
            get
            {
                Ensure.Always.NotDisposed(this);
                return _pointer;
            }
            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
        {
            get
            {
                Ensure.Always.NotDisposed(this);
                return _reservedCapacity;
            }
            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><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><para>Была выполнена попытка установить используемую емкость на
        ↪  значение, которое больше, чем зарезервированная емкость или меньше
        ↪  нуля.</para></exception>
        public long UsedCapacity
        {
            get
            {
                Ensure.Always.NotDisposed(this);
                return _usedCapacity;
            }
            set
            {
                Ensure.Always.NotDisposed(this);
                if (value != _usedCapacity)
                {
                    Ensure.Always.ArgumentInRange(value, new Range<long>(0, _reservedCapacity));
                    _usedCapacity = value;
                }
            }
        }

        #endregion

        #region DisposableBase Properties

        /// <inheritdoc/>
        protected override bool AllowMultipleDisposeCalls => true;

        #endregion
```

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

./TemporaryFileMappedResizableDirectMemory.cs

```
1   using System.IO;
2
3   namespace Platform.Memory
4   {
5       /// <summary>
6       /// <para>Represents a memory block stored as a temporary file on disk.</para>
7       /// <para>Представляет блок памяти, хранящийся в виде временного файла на диске.</para>
8       /// </summary>
9       public class TemporaryFileMappedResizableDirectMemory : FileMappedResizableDirectMemory
10      {
11          #region DisposableBase Properties
12
13          /// <inheritdoc/>
14          protected override string ObjectName => $"Temporary file stored memory block at
             ↪  '{Path}' path.";
15
16          #endregion
17
18          #region Constructors
19
20          /// <summary>
21          /// <para>Initializes a new instance of the <see
             ↪  cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
22          /// <para>Инициализирует новый экземпляр класса <see
             ↪  cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
23          /// </summary>
24          /// <param name="minimumReservedCapacity"><para>Minimum file size in
             ↪  bytes.</para><para>Минимальный размер файла в байтах.</para></param>
25          public TemporaryFileMappedResizableDirectMemory(long minimumReservedCapacity) :
             ↪  base(System.IO.Path.GetTempFileName(), minimumReservedCapacity) { }
```

```csharp
        /// <summary>
        /// <para>Initializes a new instance of the <see
        ↪  cref="TemporaryFileMappedResizableDirectMemory"/> class.</para>
        /// <para>Инициализирует новый экземпляр класса <see
        ↪  cref="TemporaryFileMappedResizableDirectMemory"/>.</para>
        /// </summary>
        public TemporaryFileMappedResizableDirectMemory() : this(MinimumCapacity) { }

        #endregion

        #region DisposableBase Methods

        /// <inheritdoc/>
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            base.Dispose(manual, wasDisposed);
            if (!wasDisposed)
            {
                File.Delete(Path);
            }
        }

        #endregion
    }
}
```

Index