

LinksPlatform's Platform.Disposables Class Library

1.1 ./Platform.Disposables/Disposable.cs

```
1 using System;
2
3 namespace Platform.Disposables
4 {
5     /// <summary>
6     /// <para>Represents disposable object that contains OnDispose event which is raised when
7     ///     ↳ the object itself is disposed.</para>
8     /// <para>Представляет высвобождаемый объект, который содержит событие OnDispose, которое
9     ///     ↳ возникает при высвобождении самого объекта.</para>
10    /// </summary>
11    public class Disposable : DisposableBase
12    {
13        private static readonly Disposal _emptyDelegate = (manual, wasDisposed) => { };
14
15        /// <summary>
16        /// <para>Occurs when the object is being disposed.</para>
17        /// <para>Возникает, когда объект высвобождается.</para>
18        /// </summary>
19        public event Disposal OnDispose;
20
21        /// <summary>
22        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
23        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
24        /// </summary>
25        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
26        ///     ↳ <see cref="Action"/>.</para></param>
27        public Disposable(Action action)
28        {
29            OnDispose = (manual, wasDisposed) =>
30            {
31                if (!wasDisposed)
32                {
33                    action();
34                }
35            };
36        }
37
38        /// <summary>
39        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
40        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
41        /// </summary>
42        /// <param name="disposal"><para>The <see cref="Disposal"/>
43        ///     ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
44        public Disposable(Disposal disposal) => OnDispose = disposal;
45
46        /// <summary>
47        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
48        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
49        /// </summary>
50        public Disposable() => OnDispose = _emptyDelegate;
51
52        /// <summary>
53        /// <para>Creates a new <see cref="Disposable"/> object initialized with specified
54        ///     ↳ delegate <see cref="Action"/>.</para>
55        /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
56        ///     ↳ указанного делегата <see cref="Action"/>.</para>
57        /// </summary>
58        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
59        ///     ↳ <see cref="Action"/>.</para></param>
60        public static implicit operator Disposable(Action action) => new Disposable(action);
61
62        /// <summary>
63        /// <para>Creates a new <see cref="Disposable"/> object initialized with specified
64        ///     ↳ delegate <see cref="Disposal"/>.</para>
65        /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
66        ///     ↳ указанного делегата <see cref="Disposal"/>.</para>
67        /// </summary>
68        /// <param name="disposal"><para>The <see cref="Disposal"/>
69        ///     ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
70        public static implicit operator Disposable(Disposal disposal) => new
71        Disposable(disposal);
72
73        /// <summary>
74        /// <para>Disposes unmanaged resources.</para>
75        /// <para>Высвобождает неуправляемые ресурсы.</para>
76    }
```

```

65     /// </summary>
66     /// <param name="manual">
67     /// <para>A value that determines whether the disposal was triggered manually (by the
    → developer's code) or was executed automatically without an explicit indication from
    → the developer.</para>
68     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    → разработчика) или же выполнилось автоматически без явного указания со стороны
    → разработчика.</para>
69     /// </param>
70     /// <param name="wasDisposed">
71     /// <para>A value that determines whether the object was released before calling this
    → method.</para>
72     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
73     /// </param>
74     protected override void Dispose(bool manual, bool wasDisposed) =>
    → RaiseOnDisposeEvent(manual, wasDisposed);
75
76     /// <summary>
77     /// <para>Raises an unmanaged resource dispose event.</para>
78     /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
79     /// </summary>
80     /// <param name="manual">
81     /// <para>A value that determines whether the disposal was triggered manually (by the
    → developer's code) or was executed automatically without an explicit indication from
    → the developer.</para>
82     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    → разработчика) или же выполнилось автоматически без явного указания со стороны
    → разработчика.</para>
83     /// </param>
84     /// <param name="wasDisposed">
85     /// <para>A value that determines whether the object was released before calling this
    → method.</para>
86     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
87     /// </param>
88     protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
    → wasDisposed);
89
90     /// <summary>
91     /// <para>Attempts to dispose the specified object, as well as set the value of the
    → variable containing this object to the default value.</para>
92     /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
    → переменной содержащей этот объект в значение по умолчанию.</para>
93     /// </summary>
94     /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
    → объекта.</para></typeparam>
95     /// <param name="object"><para>The object to dispose.</para><para>Объект, который
    → необходимо высвободить.</para></param>
96     /// <returns><para>A value that determines whether the attempt to release the specified
    → object was successful.</para><para>Значение, определяющие удачно ли была выполнена
    → попытка высвободить указанный объект.</para></returns>
97     public static bool TryDisposeAndResetToDefault<T>(ref T @object)
98     {
99         var result = @object.TryDispose();
100         if (result)
101         {
102             @object = default;
103         }
104         return result;
105     }
106 }
107 }

```

1.2 ./Platform.Disposables/DisposableBase.cs

```

1  using System;
2  using System.Collections.Concurrent;
3  using System.Threading;
4  using Platform.Exceptions;
5
6  namespace Platform.Disposables
7  {
8      /// <summary>
9      /// <para>Provides a base implementation for IDisposable interface with the basic logic
    → necessary to increase the likelihood of correct unmanaged resources release.</para>
10     /// <para>Предоставляет базовую реализацию для интерфейса IDisposable с основной логикой
    → необходимой для повышения вероятности корректного высвобождения неуправляемых
    → ресурсов.</para>
11     /// </summary>

```

```

12 public abstract class DisposableBase : IDisposable
13 {
14     private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
15     private static readonly ConcurrentStack<WeakReference<DisposableBase>>
        ↳ _disposablesWeekReferencesStack = new
        ↳ ConcurrentStack<WeakReference<DisposableBase>>();
16
17     private volatile int _disposed;
18
19     /// <summary>
20     /// <para>Gets a value indicating whether the object was disposed.</para>
21     /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
22     /// </summary>
23     public bool IsDisposed => _disposed > 0;
24
25     /// <summary>
26     /// <para>Gets the name of an object or a unique string describing this object.</para>
27     /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
28     /// </summary>
29     protected virtual string ObjectName => GetType().Name;
30
31     /// <summary>
32     /// <para>Gets a value indicating whether multiple attempts to dispose this object are
        ↳ allowed.</para>
33     /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток
        ↳ высвободить этот объект.</para>
34     /// </summary>
35     protected virtual bool AllowMultipleDisposeAttempts => false;
36
37     /// <summary>
38     /// <para>Gets a value indicating whether it is allowed to call this object disposal
        ↳ multiple times.</para>
39     /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
        ↳ высвобождение этого объекта.</para>
40     /// </summary>
41     protected virtual bool AllowMultipleDisposeCalls => false;
42
43     static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
44
45     /// <summary>
46     /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
47     /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
48     /// </summary>
49     protected DisposableBase()
50     {
51         _disposed = 0;
52         _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
53     }
54
55     /// <summary>
56     /// <para>Performs any necessary final clean-up when a class instance is being collected
        ↳ by the garbage collector.</para>
57     /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
        ↳ собирает экземпляр класса.</para>
58     /// </summary>
59     ~DisposableBase() => Destruct();
60
61     /// <summary>
62     /// <para>Disposes unmanaged resources.</para>
63     /// <para>Высвобождает неуправляемые ресурсы.</para>
64     /// </summary>
65     /// <param name="manual">
66     /// <para>A value that determines whether the disposal was triggered manually (by the
        ↳ developer's code) or was executed automatically without an explicit indication from
        ↳ the developer.</para>
67     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
        ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
        ↳ разработчика.</para>
68     /// </param>
69     /// <param name="wasDisposed">
70     /// <para>A value that determines whether the object was released before calling this
        ↳ method.</para>
71     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
72     /// </param>
73     protected abstract void Dispose(bool manual, bool wasDisposed);
74
75     /// <summary>

```

```

76    /// <para>Performs application-defined tasks associated with freeing, releasing, or
    ↪ resetting unmanaged resources.</para>
77    /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    ↪ высвобождением или сбросом неуправляемых ресурсов.</para>
78    /// </summary>
79    public void Dispose()
80    {
81        Dispose(true);
82        GC.SuppressFinalize(this);
83    }
84
85    /// <summary>
86    /// <para>Performs application-defined tasks associated with freeing, releasing, or
    ↪ resetting unmanaged resources without throwing any exceptions.</para>
87    /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    ↪ высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
88    /// </summary>
89    /// <remarks>
90    /// <para>Should be called only from classes destructors, or in case exceptions should
    ↪ be not thrown.</para>
91    /// <para>Должен вызываться только из деструкторов классов, или в случае, если
    ↪ исключения выбрасывать нельзя.</para>
92    /// </remarks>
93    public void Destruct()
94    {
95        try
96        {
97            if (!IsDisposed)
98            {
99                Dispose(false);
100            }
101        }
102        catch (Exception exception)
103        {
104            exception.Ignore();
105        }
106    }
107
108    /// <summary>
109    /// <para>Disposes unmanaged resources.</para>
110    /// <para>Высвобождает неуправляемые ресурсы.</para>
111    /// </summary>
112    /// <param name="manual">
113    /// <para>A value that determines whether the disposal was triggered manually (by the
    ↪ developer's code) or was executed automatically without an explicit indication from
    ↪ the developer.</para>
114    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↪ разработчика.</para>
115    /// </param>
116    protected virtual void Dispose(bool manual)
117    {
118        var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
119        var wasDisposed = originalDisposedValue > 0;
120        if (wasDisposed && !AllowMultipleDisposeCalls && manual)
121        {
122            Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not
            ↪ allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
123        }
124        if (AllowMultipleDisposeAttempts || !wasDisposed)
125        {
126            Dispose(manual, wasDisposed);
127        }
128    }
129
130    private static void OnProcessExit(object sender, EventArgs e)
131    {
132        while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
            ↪ weakReference))
133        {
134            if (weakReference.TryGetTarget(out DisposableBase disposable))
135            {
136                GC.SuppressFinalize(disposable);
137                disposable.Destruct();
138            }
139        }
140        UnsubscribeFromProcessExitedEventIfPossible();

```

```

141     }
142
143     private static void UnsubscribeFromProcessExitedEventIfPossible()
144     {
145         try
146         {
147             if (_currentDomain != null)
148             {
149                 _currentDomain.ProcessExit -= OnProcessExit;
150             }
151             else
152             {
153                 AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
154             }
155         }
156         catch (Exception exception)
157         {
158             exception.Ignore();
159         }
160     }
161 }
162 }

```

1.3 ./Platform.Disposables/Disposable[TPrimary, TAuxiliary].cs

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// <para>Represents disposable container that disposes two contained objects when the
7      ///     ↪ container itself is disposed.</para>
8      /// <para>Представляет высвобождаемый контейнер, который высвобождает два содержащихся в
9      ///     ↪ нём объектов при высвобождении самого контейнера.</para>
10     /// </summary>
11     /// <typeparam name="TPrimary"><para>The primary object type.</para><para>Тип основного
12     ///     ↪ объекта.</para></typeparam>
13     /// <typeparam name="TAuxiliary"><para>The auxiliary object type.</para><para>Тип
14     ///     ↪ вспомогательного объекта.</para></typeparam>
15     public class Disposable<TPrimary, TAuxiliary> : Disposable<TPrimary>
16     {
17         /// <summary>
18         /// <para>Gets the auxiliary object.</para>
19         /// <para>Возвращает вспомогательный объект.</para>
20         /// </summary>
21         public TAuxiliary AuxiliaryObject { get; }
22
23         /// <summary>
24         /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
25         ///     ↪ TAuxiliary}"> object.</para>
26         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
27         ///     ↪ TAuxiliary}">.</para>
28         /// </summary>
29         /// <param name="object"><para>The primary object.</para><para>Основной
30         ///     ↪ объект.</para></param>
31         /// <param name="auxiliaryObject"><para>The auxiliary
32         ///     ↪ object.</para><para>Вспомогательный объект.</para></param>
33         /// <param name="action"><para>The <see cref="Action{TPrimary, TAuxiliary}">
34         ///     ↪ delegate.</para><para>Делегат <see cref="Action{TPrimary,
35         ///     ↪ TAuxiliary}">.</para></param>
36         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPrimary,
37         ///     ↪ TAuxiliary> action)
38         : base(@object)
39         {
40             AuxiliaryObject = auxiliaryObject;
41             OnDispose += (manual, wasDisposed) =>
42             {
43                 if (!wasDisposed)
44                 {
45                     action(Object, AuxiliaryObject);
46                 }
47             };
48         }
49
50         /// <summary>
51         /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
52         ///     ↪ TAuxiliary}"> object.</para>
53         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
54         ///     ↪ TAuxiliary}">.</para>

```

```

42 /// </summary>
43 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
44 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
45 /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
    ↳ <see cref="Action"/>.</para></param>
46 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
    ↳ base(@object, action) => AuxiliaryObject = auxiliaryObject;
47
48 /// <summary>
49 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/> object.</para>
50 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/>.</para>
51 /// </summary>
52 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
53 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
54 /// <param name="disposal"><para>The <see cref="Disposal"/>
    ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
55 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
    ↳ base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
56
57 /// <summary>
58 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/> object.</para>
59 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/>.</para>
60 /// </summary>
61 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
62 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
63 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
    ↳ AuxiliaryObject = auxiliaryObject;
64
65 /// <summary>
66 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/> object.</para>
67 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/>.</para>
68 /// </summary>
69 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
70 public Disposable(TPrimary @object) : base(@object) { }
71
72 /// <summary>
73 /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
    ↳ initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1"/> as
    ↳ <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
    ↳ TAuxiliary, TAction}.Item2"/> as <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item3"/> as delegate <see cref="Action{TPrimary, TAuxiliary}"/>.</para>
74 /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
    ↳ инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
    ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2"/> как <see
    ↳ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
    ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3"/> как делегат <see
    ↳ cref="Action{TPrimary, TAuxiliary}"/>.</para>
75 /// </summary>
76 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
77 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
    ↳ TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
78
79 /// <summary>
80 /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
    ↳ initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1"/> as
    ↳ <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
    ↳ TAuxiliary, TAction}.Item2"/> as <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item3"/> as delegate <see cref="Action"/>.</para>

```

```

81  /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}>/>,
    инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↪ TAction}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
    ↪ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2"/> как <see
    ↪ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
    ↪ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3"/> как делегат <see
    ↪ cref="Action"/>.</para>
82  /// </summary>
83  /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
84  public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↪ TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
    ↪ tuple.Item2, tuple.Item3);
85
86  /// <summary>
87  /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}>/> object
    initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item1"/> as
    ↪ <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
    ↪ TAuxiliary, TDisposal}.Item2"/> as <see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↪ TDisposal}.Item3"/> as delegate <see cref="Disposal"/>.</para>
88  /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}>/>,
    инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↪ TDisposal}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
    ↪ cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item2"/> как <see
    ↪ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
    ↪ cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item3"/> как делегат <see
    ↪ cref="Disposal"/>.</para>
89  /// </summary>
90  /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
91  public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↪ TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
    ↪ tuple.Item2, tuple.Item3);
92
93  /// <summary>
94  /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}>/> object
    initialized with <see cref="ValueTuple{TPrimary, TAuxiliary}.Item1"/> as <see
    ↪ cref="Disposable{TPrimary}.Object"/> and <see cref="ValueTuple{TPrimary,
    ↪ TAuxiliary}.Item2"/> as <see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/>.</para>
95  /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}>/>,
    инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary}.Item1"/>
    ↪ как <see cref="Disposable{TPrimary}.Object"/> и <see cref="ValueTuple{TPrimary,
    ↪ TAuxiliary}.Item2"/> как <see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/>.</para>
96  /// </summary>
97  /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
98  public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↪ TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
99
100  /// <summary>
101  /// <para>Creates a new copy of the primary object (<see
    ↪ cref="Disposable{TPrimary}.Object"/>).</para>
102  /// <para>Создаёт новую копию основного объекта (<see
    ↪ cref="Disposable{TPrimary}.Object"/>).</para>
103  /// </summary>
104  /// <param name="disposableContainer"><para>The disposable
    ↪ container.</para><para>Высвобождаемый контейнер.</para></param>
105  public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
    ↪ disposableContainer) => disposableContainer.Object;
106
107  /// <summary>
108  /// <para>Creates a new copy of the auxiliary object (<see
    ↪ cref="Disposable{TPrimary}.Object"/>).</para>
109  /// <para>Создаёт новую копию вспомогательного объекта (<see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/>).</para>
110  /// </summary>
111  /// <param name="disposableContainer"><para>The disposable
    ↪ container.</para><para>Высвобождаемый контейнер.</para></param>
112  public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
    ↪ disposableContainer) => disposableContainer.AuxiliaryObject;
113
114  /// <summary>
115  /// <para>Disposes unmanaged resources.</para>
116  /// <para>Высвобождает неуправляемые ресурсы.</para>
117  /// </summary>

```

```

118     /// <param name="manual">
119     /// <para>A value that determines whether the disposal was triggered manually (by the
    ↳ developer's code) or was executed automatically without an explicit indication from
    ↳ the developer.</para>
120     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↳ разработчика.</para>
121     /// </param>
122     /// <param name="wasDisposed">
123     /// <para>A value that determines whether the object was released before calling this
    ↳ method.</para>
124     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
125     /// </param>
126     protected override void Dispose(bool manual, bool wasDisposed)
127     {
128         RaiseOnDisposeEvent(manual, wasDisposed);
129         AuxiliaryObject.TryDispose();
130         Object.TryDispose();
131     }
132 }
133 }

```

1.4 ./Platform.Disposables/Disposable[T].cs

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// <para>Represents disposable container that disposes contained object when the container
    ↳ itself is disposed.</para>
7      /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
    ↳ объект при высвобождении самого контейнера.</para>
8      /// </summary>
9      public class Disposable<T> : Disposable
10     {
11         /// <summary>
12         /// <para>Gets the object.</para>
13         /// <para>Возвращает объект.</para>
14         /// </summary>
15         public T Object { get; }
16
17         /// <summary>
18         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
19         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
20         /// </summary>
21         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
22         /// <param name="action"><para>The <see cref="Action{T}"> delegate.</para><para>Делегат
    ↳ <see cref="Action{T}">.</para></param>
23         public Disposable(T @object, Action<T> action)
24         {
25             Object = @object;
26             OnDispose += (manual, wasDisposed) =>
27             {
28                 if (!wasDisposed)
29                 {
30                     action(Object);
31                 }
32             };
33         }
34
35         /// <summary>
36         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
37         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
38         /// </summary>
39         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
40         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
    ↳ <see cref="Action"/>.</para></param>
41         public Disposable(T @object, Action action) : base(action) => Object = @object;
42
43         /// <summary>
44         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
45         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
46         /// </summary>
47         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
48         /// <param name="disposal"><para>The <see cref="Disposal"/>
    ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
49         public Disposable(T @object, Disposal disposal) : base(disposal) => Object = @object;
50     }

```



```

51 /// <summary>
52 /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
53 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
54 /// </summary>
55 /// <param name="object"><para>The object.</para><para>Объект.</para></param>
56 public Disposable(T @object) => Object = @object;
57
58 /// <summary>
59 /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with <see
    ↳ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
    ↳ <see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see
    ↳ cref="Action{T}"/>.</para>
60 /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
    ↳ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
    ↳ и <see cref="ValueTuple{T, TAction}.Item2"/> как делегат <see
    ↳ cref="Action{T}"/>.</para>
61 /// </summary>
62 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
63 public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
    ↳ Disposable<T>(tuple.Item1, tuple.Item2);
64
65 /// <summary>
66 /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with <see
    ↳ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
    ↳ <see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see cref="Action"/>.</para>
67 /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
    ↳ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
    ↳ и <see cref="ValueTuple{T, TAction}.Item2"/> как делегат <see cref="Action"/>.</para>
68 /// </summary>
69 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
70 public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
    ↳ Disposable<T>(tuple.Item1, tuple.Item2);
71
72 /// <summary>
73 /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with <see
    ↳ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
    ↳ <see cref="ValueTuple{T, TDisposal}.Item2"/> as delegate <see
    ↳ cref="Disposal"/>.</para>
74 /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
    ↳ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
    ↳ и <see cref="ValueTuple{T, TDisposal}.Item2"/> как делегат <see
    ↳ cref="Disposal"/>.</para>
75 /// </summary>
76 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
77 public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
    ↳ Disposable<T>(tuple.Item1, tuple.Item2);
78
79 /// <summary>
80 /// <para>Creates a new <see cref="Disposable{T}"/> object initialized with specified
    ↳ object as <see cref="Disposable{T}.Object"/>.</para>
81 /// <para>Создает новый объект <see cref="Disposable{T}"/>, инициализированную с помощью
    ↳ указанного объекта как <see cref="Disposable{T}.Object"/>.</para>
82 /// </summary>
83 /// <param name="object"><para>The object.</para><para>Объект.</para></param>
84 public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
85
86 /// <summary>
87 /// <para>Creates a new copy of the primary object (<see
    ↳ cref="Disposable{T}.Object"/>).</para>
88 /// <para>Создаёт новую копию основного объекта (<see
    ↳ cref="Disposable{T}.Object"/>).</para>
89 /// </summary>
90 /// <param name="disposableContainer"><para>The disposable
    ↳ container.</para><para>Высвобождаемый контейнер.</para></param>
91 public static implicit operator T(Disposable<T> disposableContainer) =>
    ↳ disposableContainer.Object;
92
93 /// <summary>
94 /// <para>Disposes unmanaged resources.</para>
95 /// <para>Высвобождает неуправляемые ресурсы.</para>
96 /// </summary>
97 /// <param name="manual">
98 /// <para>A value that determines whether the disposal was triggered manually (by the
    ↳ developer's code) or was executed automatically without an explicit indication from
    ↳ the developer.</para>

```

```

99     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    → разработчика) или же выполнилось автоматически без явного указания со стороны
    → разработчика.</para>
100    /// </param>
101    /// <param name="wasDisposed">
102    /// <para>A value that determines whether the object was released before calling this
    → method.</para>
103    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
104    /// </param>
105    protected override void Dispose(bool manual, bool wasDisposed)
106    {
107        base.Dispose(manual, wasDisposed);
108        Object.TryDispose();
109    }
110 }
111 }

```

1.5 ./Platform.Disposables/Disposal.cs

```

1 namespace Platform.Disposables
2 {
3     /// <summary>
4     /// <para>Encapsulates a method that is used to dispose unmanaged resources.</para>
5     /// <para>Инкапсулирует метод, который используется для высвобождения неуправляемых
    → ресурсов.</para>
6     /// </summary>
7     /// <param name="manual">
8     /// <para>A value that determines whether the disposal was triggered manually (by the
    → developer's code) or was executed automatically without an explicit indication from the
    → developer.</para>
9     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом разработчика)
    → или же выполнилось автоматически без явного указания со стороны разработчика.</para>
10    /// </param>
11    /// <param name="wasDisposed">
12    /// <para>A value that determines whether the object was released before calling this
    → method.</para>
13    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
14    /// </param>
15    public delegate void Disposal(bool manual, bool wasDisposed);
16 }

```

1.6 ./Platform.Disposables/EnsureExtensions.cs

```

1 using System;
2 using System.Diagnostics;
3 using System.Runtime.CompilerServices;
4 using Platform.Exceptions;
5 using Platform.Exceptions.ExtensionRoots;
6
7 #pragma warning disable IDE0060 // Remove unused parameter
8
9 namespace Platform.Disposables
10 {
11     /// <summary>
12     /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/>
    → and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
13     /// <para>Предоставляет набор методов расширения для объектов <see
    → cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
14     /// </summary>
15     public static class EnsureExtensions
16     {
17         #region Always
18
19         /// <summary>
20         /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
    → has not been released. This check is performed regardless of the build
    → configuration.</para>
21         /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
    → был высвобожден. Эта проверка выполняется независимо от конфигурации
    → сборки.</para>
22         /// </summary>
23         /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
24         /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
    → interface.</para><para>Объект, реализующий интерфейс <see
    → cref="IDisposable"/></para></param>
25         /// <param name="objectName"><para>The name of object.</para><para>Имя
    → объекта.</para></param>
26         /// <param name="message"><para>The message of the thrown
    → exception.</para><para>Сообщение выбрасываемого исключения.</para></param>

```

```

27 [MethodImpl(MethodImplOptions.AggressiveInlining)]
28 public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
    ↳ disposable, string objectName, string message)
29 {
30     if (disposable.IsDisposed)
31     {
32         throw new ObjectDisposedException(objectName, message);
33     }
34 }
35
36 /// <summary>
37 /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
    ↳ has not been released. This check is performed regardless of the build
    ↳ configuration.</para>
38 /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
    ↳ был высвобожден. Эта проверка выполняется независимо от конфигурации
    ↳ сборки.</para>
39 /// </summary>
40 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
41 /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
    ↳ interface.</para><para>Объект, реализующий интерфейс <see
    ↳ cref="IDisposable"/></para></param>
42 /// <param name="objectName"><para>The name of object.</para><para>Имя
    ↳ объекта.</para></param>
43 [MethodImpl(MethodImplOptions.AggressiveInlining)]
44 public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
    ↳ disposable, string objectName) => NotDisposed(root, disposable, objectName, null);
45
46 /// <summary>
47 /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
    ↳ has not been released. This check is performed regardless of the build
    ↳ configuration.</para>
48 /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
    ↳ был высвобожден. Эта проверка выполняется независимо от конфигурации
    ↳ сборки.</para>
49 /// </summary>
50 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
51 /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
    ↳ interface.</para><para>Объект, реализующий интерфейс <see
    ↳ cref="IDisposable"/></para></param>
52 [MethodImpl(MethodImplOptions.AggressiveInlining)]
53 public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
    ↳ disposable) => NotDisposed(root, disposable, null, null);
54
55 #endregion
56
57 #region OnDebug
58
59 /// <summary>
60 /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
    ↳ has not been released. This check is performed only for DEBUG build
    ↳ configuration.</para>
61 /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
    ↳ был высвобожден. Эта проверка выполняется только для конфигурации сборки
    ↳ DEBUG.</para>
62 /// </summary>
63 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
64 /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
    ↳ interface.</para><para>Объект, реализующий интерфейс <see
    ↳ cref="IDisposable"/></para></param>
65 /// <param name="objectName"><para>The name of object.</para><para>Имя
    ↳ объекта.</para></param>
66 /// <param name="message"><para>The message of the thrown
    ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
67 [Conditional("DEBUG")]
68 public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
    ↳ disposable, string objectName, string message) =>
    ↳ Ensure.Always.NotDisposed(disposable, objectName, message);
69
70 /// <summary>
71 /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
    ↳ has not been released. This check is performed only for DEBUG build
    ↳ configuration.</para>

```

```

72    /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
    → был высвобожден. Эта проверка выполняется только для конфигурации сборки
    → DEBUG.</para>
73    /// </summary>
74    /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
75    /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
    → interface.</para><para>Объект, реализующий интерфейс <see
    → cref="IDisposable"/></para></param>
76    /// <param name="objectName"><para>The name of object.</para><para>Имя
    → объекта.</para></param>
77    [Conditional("DEBUG")]
78    public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
    → disposable, string objectName) => Ensure.Always.NotDisposed(disposable, objectName,
    → null);
79
80    /// <summary>
81    /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
    → has not been released. This check is performed only for DEBUG build
    → configuration.</para>
82    /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
    → был высвобожден. Эта проверка выполняется только для конфигурации сборки
    → DEBUG.</para>
83    /// </summary>
84    /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
85    /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
    → interface.</para><para>Объект, реализующий интерфейс <see
    → cref="IDisposable"/></para></param>
86    [Conditional("DEBUG")]
87    public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
    → disposable) => Ensure.Always.NotDisposed(disposable, null, null);
88
89    #endregion
90 }
91 }

```

1.7 ./Platform.Disposables/GenericObjectExtensions.cs

```

1  using System;
2  using Platform.Exceptions;
3
4  namespace Platform.Disposables
5  {
6      /// <summary>
7      /// <para>Provides a set of static methods that help dispose a generic objects.</para>
8      /// <para>Предоставляет набор статических методов которые помогают высвободить универсальные
    → объекты.</para>
9      /// </summary>
10     static public class GenericObjectExtensions
11     {
12         /// <summary>
13         /// <para>Attempts to dispose the specified object.</para>
14         /// <para>Выполняет попытку высвободить указанный объект.</para>
15         /// </summary>
16         /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
    → объекта.</para></typeparam>
17         /// <param name="object"><para>The object to dispose.</para><para>Объект, который
    → необходимо высвободить.</para></param>
18         /// <returns><para>A value that determines whether the attempt to release the specified
    → object was successful.</para><para>Значение, определяющие удачно ли была выполнена
    → попытка высвободить указанный объект.</para></returns>
19         public static bool TryDispose<T>(this T @object)
20         {
21             try
22             {
23                 if (@object is DisposableBase disposableBase)
24                 {
25                     disposableBase.DisposeIfNotDisposed();
26                 }
27                 else if (@object is System.IDisposable disposable)
28                 {
29                     disposable.Dispose();
30                 }
31                 return true;
32             }
33             catch (Exception exception)
34             {

```

```

35         exception.Ignore();
36     }
37     return false;
38 }
39
40 /// <summary>
41 /// <para>Attempts to dispose the specified object.</para>
42 /// <para>Выполняет попытку высвободить указанный объект.</para>
43 /// </summary>
44 /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
    ↳ объекта.</para></typeparam>
45 /// <param name="object"><para>The object to dispose.</para><para>Объект, который
    ↳ необходимо высвободить.</para></param>
46 public static void DisposeIfPossible<T>(this T @object) => TryDispose(@object);
47 }
48 }

```

1.8 ./Platform.Disposables/IDisposable.cs

```

1 namespace Platform.Disposables
2 {
3     /// <summary>
4     /// <para>Представляет расширенный интерфейс <see cref="System.IDisposable"/>.</para>
5     /// <para>Represents an extended <see cref="System.IDisposable"/> interface.</para>
6     /// </summary>
7     public interface IDisposable : System.IDisposable
8     {
9         /// <summary>
10        /// <para>Gets a value indicating whether the object was disposed.</para>
11        /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
12        /// </summary>
13        bool IsDisposed { get; }
14
15        /// <summary>
16        /// <para>Performs application-defined tasks associated with freeing, releasing, or
17        ↳ resetting unmanaged resources without throwing any exceptions.</para>
18        /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
19        ↳ высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
20        /// </summary>
21        /// <remarks>
22        /// <para>Should be called only from classes destructors, or in case exceptions should
23        ↳ be not thrown.</para>
24        /// <para>Должен вызываться только из деструкторов классов, или в случае, если
25        ↳ исключения выбрасывать нельзя.</para>
26        /// </remarks>
27        void Destruct();
28    }
29 }

```

1.9 ./Platform.Disposables/IDisposableExtensions.cs

```

1 namespace Platform.Disposables
2 {
3     /// <summary>
4     /// <para>Provides a set of extension methods for <see cref="IDisposable"/> objects.</para>
5     /// <para>Предоставляет набор методов расширения для объектов <see
6     ↳ cref="IDisposable"/>.</para>
7     /// </summary>
8     public static class IDisposableExtensions
9     {
10        /// <summary>
11        /// <para>Attempts to dispose the specified object.</para>
12        /// <para>Выполняет попытку высвободить указанный объект.</para>
13        /// </summary>
14        /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
15        ↳ interface.</para><para>Объект, реализующий интерфейс <see
16        ↳ cref="IDisposable"/></para></param>
17        public static void DisposeIfNotDisposed(this IDisposable disposable)
18        {
19            if (!disposable.IsDisposed)
20            {
21                disposable.Dispose();
22            }
23        }
24    }
25 }

```

1.10 ./Platform.Disposables.Tests/DisposableTests.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.IO;
5  using System.Linq;
6  using System.Threading;
7  using Xunit;
8
9  namespace Platform.Disposables.Tests
10 {
11     public static class DisposableTests
12     {
13         [Fact]
14         public static void DisposalOrderTest()
15         {
16             var logPath = Path.GetTempFileName();
17             using (var process = Process.Start(CreateProcessStartInfo(logPath,
18                 ↪ waitForCancellation: false)))
19             {
20                 process.WaitForExit();
21             }
22             var result = File.ReadAllText(logPath);
23             Assert.Equal("21", result);
24             File.Delete(logPath);
25         }
26
27         [Fact]
28         public static void DisposalAtProcessKillTest()
29         {
30             var logPath = Path.GetTempFileName();
31             using (var process = Process.Start(CreateProcessStartInfo(logPath,
32                 ↪ waitForCancellation: true)))
33             {
34                 Thread.Sleep(1000);
35                 process.Kill();
36             }
37             var result = File.ReadAllText(logPath);
38             Assert.Equal("", result); // Currently, process termination will not release
39                                     ↪ resources
40             File.Delete(logPath);
41         }
42
43         private static ProcessStartInfo CreateProcessStartInfo(string logPath, bool
44             ↪ waitForCancellation)
45         {
46             var projectPath = GetDisposalObjectTestProjectFilePath();
47             return new ProcessStartInfo
48             {
49                 FileName = "dotnet",
50                 Arguments = ["run -p \"{projectPath}\" -f netcoreapp2.1 \"{logPath}\"",
51                     ↪ {waitForCancellation.ToString()}"],
52                 UseShellExecute = false,
53                 CreateNoWindow = true
54             };
55         }
56
57         private static string GetDisposalObjectTestProjectFilePath()
58         {
59             const string currentProjectName = nameof(Platform) + "." + nameof(Disposables) + "." +
60                 ↪ + nameof(Tests);
61             const string disposalOrderTestProjectName = currentProjectName + "." +
62                 ↪ + nameof(DisposalOrderTest);
63             var currentDirectory = Environment.CurrentDirectory;
64             var pathParts = currentDirectory.Split(Path.DirectorySeparatorChar);
65             var newPathParts = new List<string>();
66             for (var i = 0; i < pathParts.Length; i++)
67             {
68                 if (string.Equals(pathParts[i], currentProjectName))
69                 {
70                     newPathParts.Add(disposalOrderTestProjectName);
71                     break;
72                 }
73                 else
74                 {
75                     newPathParts.Add(pathParts[i]);
76                 }
77             }
78             pathParts = newPathParts.ToArray();

```

```

72 #if NET471
73     var directory = string.Join(Path.DirectorySeparatorChar.ToString(),
74         ↪ pathParts.ToArray());
75 #else
76     var directory = Path.Combine(pathParts);
77 #endif
78     var path = Path.Combine(directory, $"{disposalOrderTestProjectName}.csproj");
79     if (!Path.IsPathRooted(path))
80     {
81         path = $"{Path.DirectorySeparatorChar}{path}";
82     }
83     return path;
84 }
85 }

```

1.11 ./Platform.Disposables.Tests/SystemTests.cs

```

1  using Xunit;
2
3  namespace Platform.Disposables.Tests
4  {
5      /// <summary>
6      /// <para>Contains tests for features of .NET Framework itself, that are required by current
7      ↪ implementations.</para>
8      /// <para>Содержит тесты для функций самого .NET Framework, которые требуются для текущих
9      ↪ реализаций.</para>
10     /// </summary>
11     public static class SystemTests
12     {
13         [Fact]
14         public static void UsingSupportsNullTest()
15         {
16             using var disposable = null as IDisposable;
17             Assert.True(disposable == null);
18         }
19     }
20 }

```

Index

- ./Platform.Disposables.Tests/DisposableTests.cs, 13
- ./Platform.Disposables.Tests/SystemTests.cs, 15
- ./Platform.Disposables/Disposable.cs, 1
- ./Platform.Disposables/DisposableBase.cs, 2
- ./Platform.Disposables/Disposable[TPrimary, TAuxiliary].cs, 5
- ./Platform.Disposables/Disposable[T].cs, 8
- ./Platform.Disposables/Disposal.cs, 10
- ./Platform.Disposables/EnsureExtensions.cs, 10
- ./Platform.Disposables/GenericObjectExtensions.cs, 12
- ./Platform.Disposables/IDisposable.cs, 13
- ./Platform.Disposables/IDisposableExtensions.cs, 13