

LinksPlatform's Platform.Disposables Class Library

1.1 ./csharp/Platform.Disposables/Disposable.cs

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

```

68    /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
69    → указанного делегата <see cref="Disposal"/>.</para>
70    /// </summary>
71    /// <param name="disposal"><para>The <see cref="Disposal"/>
72    → delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    public static implicit operator Disposable(Disposal disposal) => new
75    → Disposable(disposal);
76
77    /// <summary>
78    /// <para>Disposes unmanaged resources.</para>
79    /// <para>Высвобождает неуправляемые ресурсы.</para>
80    /// </summary>
81    /// <param name="manual">
82    /// <para>A value that determines whether the disposal was triggered manually (by the
83    → developer's code) or was executed automatically without an explicit indication from
84    → a developer.</para>
85    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
86    → разработчика) или же выполнилось автоматически без явного указания со стороны
87    → разработчика.</para>
88    /// </param>
89    /// <param name="wasDisposed">
90    /// <para>A value that determines whether the object was released before calling this
91    → method.</para>
92    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
93    /// </param>
94    [MethodImpl(MethodImplOptions.AggressiveInlining)]
95    protected override void Dispose(bool manual, bool wasDisposed) =>
96    → RaiseOnDisposeEvent(manual, wasDisposed);
97
98    /// <summary>
99    /// <para>Raises an unmanaged resource dispose event.</para>
100    /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
101    /// </summary>
102    /// <param name="manual">
103    /// <para>A value that determines whether the disposal was triggered manually (by the
104    → developer's code) or was executed automatically without an explicit indication from
105    → a developer.</para>
106    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
107    → разработчика) или же выполнилось автоматически без явного указания со стороны
108    → разработчика.</para>
109    /// </param>
110    /// <param name="wasDisposed">
111    /// <para>A value that determines whether the object was released before calling this
112    → method.</para>
113    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
114    /// </param>
115    [MethodImpl(MethodImplOptions.AggressiveInlining)]
116    protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
117    → wasDisposed);
118
119    /// <summary>
120    /// <para>Attempts to dispose the specified object, as well as set the value of the
121    → variable containing this object to the default value.</para>
122    /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
123    → переменной содержащей этот объект в значение по умолчанию.</para>
124    /// </summary>
125    /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
126    → объекта.</para></typeparam>
127    /// <param name="object"><para>The object to dispose.</para><para>Объект, который
128    → необходимо высвободить.</para></param>
129    /// <returns><para>A value that determines whether the attempt to release the specified
130    → object was successful.</para><para>Значение, определяющие удачно ли была выполнена
131    → попытка высвободить указанный объект.</para></returns>
132    [MethodImpl(MethodImplOptions.AggressiveInlining)]
133    public static bool TryDisposeAndResetToDefault<T>(ref T @object)
134    {
135        var result = @object.TryDispose();
136        if (result)
137        {
138            @object = default;
139        }
140        return result;
141    }
142 }

```

1.2 ./csharp/Platform.Disposables/DisposableBase.cs

```
1 using System;
2 using System.Collections.Concurrent;
3 using System.Runtime.CompilerServices;
4 using System.Threading;
5 using Platform.Exceptions;
6
7 namespace Platform.Disposables
8 {
9     /// <summary>
10    /// <para>Represents a base implementation for <see
11    ↪ cref="Platform.Disposables.IDisposable"/> interface with the basic logic necessary to
12    ↪ increase the likelihood of correct unmanaged resources release.</para>
13    /// <para>Представляет базовую реализацию для интерфейса <see
14    ↪ cref="Platform.Disposables.IDisposable"/> с основной логикой необходимой для повышения
15    ↪ вероятности корректного высвобождения неуправляемых ресурсов.</para>
16    /// </summary>
17    public abstract class DisposableBase : IDisposable
18    {
19        /// <summary>
20        /// <para>
21        /// The current domain.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
26        /// <summary>
27        /// <para>
28        /// The disposable base.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        private static readonly ConcurrentStack<WeakReference<DisposableBase>>
33        ↪ _disposablesWeakReferencesStack = new
34        ↪ ConcurrentStack<WeakReference<DisposableBase>>();
35
36        /// <summary>
37        /// <para>
38        /// The disposed.
39        /// </para>
40        /// <para></para>
41        /// </summary>
42        private volatile int _disposed;
43
44        /// <summary>
45        /// <para>Gets a value indicating whether the object was disposed.</para>
46        /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
47        /// </summary>
48        public bool IsDisposed
49        {
50            [MethodImpl(MethodImplOptions.AggressiveInlining)]
51            get => _disposed > 0;
52        }
53
54        /// <summary>
55        /// <para>Gets the name of an object or a unique string describing this object.</para>
56        /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
57        /// </summary>
58        protected virtual string ObjectName
59        {
60            [MethodImpl(MethodImplOptions.AggressiveInlining)]
61            get => GetType().Name;
62        }
63
64        /// <summary>
65        /// <para>Gets a value indicating whether multiple attempts to dispose this object are
66        ↪ allowed.</para>
67        /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток
68        ↪ высвободить этот объект.</para>
69        /// </summary>
70        protected virtual bool AllowMultipleDisposeAttempts
71        {
72            [MethodImpl(MethodImplOptions.AggressiveInlining)]
73            get => false;
74        }
75
76        /// <summary>
77        /// <para>Gets a value indicating whether it is allowed to call this object disposal
78        ↪ multiple times.</para>
```

```

70    /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
    ↪ высвобождение этого объекта.</para>
71    /// </summary>
72    protected virtual bool AllowMultipleDisposeCalls
73    {
74        [MethodImpl(MethodImplOptions.AggressiveInlining)]
75        get => false;
76    }
77
78    /// <summary>
79    /// <para>
80    /// Initializes a new <see cref="DisposableBase"/> instance.
81    /// </para>
82    /// <para></para>
83    /// </summary>
84    [MethodImpl(MethodImplOptions.AggressiveInlining)]
85    static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
86
87    /// <summary>
88    /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
89    /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
90    /// </summary>
91    [MethodImpl(MethodImplOptions.AggressiveInlining)]
92    protected DisposableBase()
93    {
94        _disposed = 0;
95        _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
96    }
97
98    /// <summary>
99    /// <para>Performs any necessary final clean-up when a class instance is being collected
    ↪ by the garbage collector.</para>
100    /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
    ↪ собирает экземпляр класса.</para>
101    /// </summary>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    ~DisposableBase() => Destruct();
104
105    /// <summary>
106    /// <para>Disposes unmanaged resources.</para>
107    /// <para>Высвобождает неуправляемые ресурсы.</para>
108    /// </summary>
109    /// <param name="manual">
110    /// <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
    ↪ a developer.</para>
111    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↪ разработчика.</para>
112    /// </param>
113    /// <param name="wasDisposed">
114    /// <para>A value that determines whether the object was released before calling this
    ↪ method.</para>
115    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
116    /// </param>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    protected abstract void Dispose(bool manual, bool wasDisposed);
119
120    /// <summary>
121    /// <para>Performs application-defined tasks associated with freeing, releasing, or
    ↪ resetting unmanaged resources.</para>
122    /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    ↪ высвобождением или сбросом неуправляемых ресурсов.</para>
123    /// </summary>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    public void Dispose()
126    {
127        Dispose(true);
128        GC.SuppressFinalize(this);
129    }
130
131    /// <summary>
132    /// <para>Performs application-defined tasks associated with freeing, releasing, or
    ↪ resetting unmanaged resources without throwing any exceptions.</para>
133    /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    ↪ высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
134    /// </summary>

```

```

135 /// <remarks>
136 /// <para>Should be called only from classes destructors, or in case exceptions should
    ↳ be not thrown.</para>
137 /// <para>Должен вызываться только из деструкторов классов, или в случае, если
    ↳ исключения выбрасывать нельзя.</para>
138 /// </remarks>
139 [MethodImpl(MethodImplOptions.AggressiveInlining)]
140 public void Destruct()
141 {
142     try
143     {
144         if (!IsDisposed)
145         {
146             Dispose(false);
147         }
148     }
149     catch (Exception exception)
150     {
151         exception.Ignore();
152     }
153 }
154
155 /// <summary>
156 /// <para>Disposes unmanaged resources.</para>
157 /// <para>Высвобождает неуправляемые ресурсы.</para>
158 /// </summary>
159 /// <param name="manual">
160 /// <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
    ↳ a developer.</para>
161 /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↳ разработчика.</para>
162 /// </param>
163 protected virtual void Dispose(bool manual)
164 {
165     var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
166     var wasDisposed = originalDisposedValue > 0;
167     if (wasDisposed && !AllowMultipleDisposeCalls && manual)
168     {
169         Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not
            ↳ allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
170     }
171     if (AllowMultipleDisposeAttempts || !wasDisposed)
172     {
173         Dispose(manual, wasDisposed);
174     }
175 }
176
177 /// <summary>
178 /// <para>
179 /// Ons the process exit using the specified sender.
180 /// </para>
181 /// <para></para>
182 /// </summary>
183 /// <param name="sender">
184 /// <para>The sender.</para>
185 /// <para></para>
186 /// </param>
187 /// <param name="e">
188 /// <para>The .</para>
189 /// <para></para>
190 /// </param>
191 private static void OnProcessExit(object sender, EventArgs e)
192 {
193     while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
        ↳ weakReference))
194     {
195         if (weakReference.TryGetTarget(out DisposableBase disposable))
196         {
197             GC.SuppressFinalize(disposable);
198             disposable.Destruct();
199         }
200     }
201     UnsubscribeFromProcessExitedEventIfPossible();
202 }
203

```

```

204     /// <summary>
205     /// <para>
206     /// Unsubscribes the from process exited event if possible.
207     /// </para>
208     /// <para></para>
209     /// </summary>
210     private static void UnsubscribeFromProcessExitedEventIfPossible()
211     {
212         try
213         {
214             if (_currentDomain != null)
215             {
216                 _currentDomain.ProcessExit -= OnProcessExit;
217             }
218             else
219             {
220                 AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
221             }
222         }
223         catch (Exception exception)
224         {
225             exception.Ignore();
226         }
227     }
228 }
229 }

```

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

```

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

```

```

40         action(Object, AuxiliaryObject);
41     }
42 };
43 }
44
45 /// <summary>
46 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
47   ↳ TAuxiliary}" /> object.</para>
48 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
49   ↳ TAuxiliary}" />.</para>
50 /// </summary>
51 /// <param name="object"><para>The primary object.</para><para>Основной
52   ↳ объект.</para></param>
53 /// <param name="auxiliaryObject"><para>The auxiliary
54   ↳ object.</para><para>Вспомогательный объект.</para></param>
55 /// <param name="action"><para>The <see cref="Action" /> delegate.</para><para>Делегат
56   ↳ <see cref="Action" />.</para></param>
57 [MethodImpl(MethodImplOptions.AggressiveInlining)]
58 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
59   ↳ base(@object, action) => AuxiliaryObject = auxiliaryObject;
60
61 /// <summary>
62 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
63   ↳ TAuxiliary}" /> object.</para>
64 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
65   ↳ TAuxiliary}" />.</para>
66 /// </summary>
67 /// <param name="object"><para>The primary object.</para><para>Основной
68   ↳ объект.</para></param>
69 /// <param name="auxiliaryObject"><para>The auxiliary
70   ↳ object.</para><para>Вспомогательный объект.</para></param>
71 /// <param name="disposal"><para>The <see cref="Disposal" />
72   ↳ delegate.</para><para>Делегат <see cref="Disposal" />.</para></param>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
75   ↳ base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
76
77 /// <summary>
78 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
79   ↳ TAuxiliary}" /> object.</para>
80 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
81   ↳ TAuxiliary}" />.</para>
82 /// </summary>
83 /// <param name="object"><para>The primary object.</para><para>Основной
84   ↳ объект.</para></param>
85 /// <param name="auxiliaryObject"><para>The auxiliary
86   ↳ object.</para><para>Вспомогательный объект.</para></param>
87 [MethodImpl(MethodImplOptions.AggressiveInlining)]
88 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
89   ↳ AuxiliaryObject = auxiliaryObject;
90
91 /// <summary>
92 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
93   ↳ TAuxiliary}" /> object.</para>
94 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
95   ↳ TAuxiliary}" />.</para>
96 /// </summary>
97 /// <param name="object"><para>The primary object.</para><para>Основной
98   ↳ объект.</para></param>
99 [MethodImpl(MethodImplOptions.AggressiveInlining)]
100 public Disposable(TPrimary @object) : base(@object) { }
101
102 /// <summary>
103 /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}" /> object
104   ↳ initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1" /> as
105   ↳ <see cref="Disposable{TPrimary}.Object" />, <see cref="ValueTuple{TPrimary,
106   ↳ TAuxiliary, TAction}.Item2" /> as <see cref="Disposable{TPrimary,
107   ↳ TAuxiliary}.AuxiliaryObject" /> and <see cref="ValueTuple{TPrimary, TAuxiliary,
108   ↳ TAction}.Item3" /> as delegate <see cref="Action{TPrimary, TAuxiliary}" />.</para>
109 /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}" />,
110   ↳ инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
111   ↳ TAction}.Item1" /> как <see cref="Disposable{TPrimary}.Object" />, <see
112   ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2" /> как <see
113   ↳ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject" /> и <see
114   ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3" /> как делегат <see
115   ↳ cref="Action{TPrimary, TAuxiliary}" />.</para>

```

```

85 /// </summary>
86 /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
87 [MethodImpl(MethodImplOptions.AggressiveInlining)]
88 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
    ↳ TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);

89
90 /// <summary>
91 /// <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>
92 /// <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>
93 /// </summary>
94 /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
95 [MethodImpl(MethodImplOptions.AggressiveInlining)]
96 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
    ↳ tuple.Item2, tuple.Item3);

97
98 /// <summary>
99 /// <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>
100 /// <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>
101 /// </summary>
102 /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
104 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
    ↳ tuple.Item2, tuple.Item3);

105
106 /// <summary>
107 /// <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>
108 /// <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>
109 /// </summary>
110 /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
111 [MethodImpl(MethodImplOptions.AggressiveInlining)]
112 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);

113
114 /// <summary>
115 /// <para>Creates a new copy of the primary object (<see
    ↳ cref="Disposable{TPrimary}.Object"/>).</para>
116 /// <para>Создаёт новую копию основного объекта (<see
    ↳ cref="Disposable{TPrimary}.Object"/>).</para>
117 /// </summary>
118 /// <param name="disposableContainer"><para>The disposable
    ↳ container.</para><para>Высвобождаемый контейнер.</para></param>
119 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

120 public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
    ↳ disposableContainer) => disposableContainer.Object;
121
122 /// <summary>
123 /// <para>Creates a new copy of the auxiliary object (<see
    ↳ cref="Disposable{TPrimary}.Object"/>).</para>
124 /// <para>Создаёт новую копию вспомогательного объекта (<see cref="Disposable{TPrimary,
    ↳ TAuxiliary}.AuxiliaryObject"/>).</para>
125 /// </summary>
126 /// <param name="disposableContainer"><para>The disposable
    ↳ container.</para><para>Высвобождаемый контейнер.</para></param>
127 [MethodImpl(MethodImplOptions.AggressiveInlining)]
128 public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
    ↳ disposableContainer) => disposableContainer.AuxiliaryObject;
129
130 /// <summary>
131 /// <para>Disposes unmanaged resources.</para>
132 /// <para>Высвобождает неуправляемые ресурсы.</para>
133 /// </summary>
134 /// <param name="manual">
135 /// <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
    ↳ a developer.</para>
136 /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↳ разработчика.</para>
137 /// </param>
138 /// <param name="wasDisposed">
139 /// <para>A value that determines whether the object was released before calling this
    ↳ method.</para>
140 /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
141 /// </param>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 protected override void Dispose(bool manual, bool wasDisposed)
144 {
145     RaiseOnDisposeEvent(manual, wasDisposed);
146     AuxiliaryObject.TryDispose();
147     Object.TryDispose();
148 }
149 }
150 }

```

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

```

1 using System;
2 using System.Runtime.CompilerServices;
3
4 namespace Platform.Disposables
5 {
6     /// <summary>
7     /// <para>Represents disposable container that disposes contained object when the container
    ↳ itself is disposed.</para>
8     /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
    ↳ объект при высвобождении самого контейнера.</para>
9     /// </summary>
10 public class Disposable<T> : Disposable
11 {
12     /// <summary>
13     /// <para>Gets the object.</para>
14     /// <para>Возвращает объект.</para>
15     /// </summary>
16 public T Object
17 {
18     [MethodImpl(MethodImplOptions.AggressiveInlining)]
19     get;
20 }
21
22 /// <summary>
23 /// <para>Initializes a new instance of the <see cref="Disposable{T}"/> object.</para>
24 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}"/>.</para>
25 /// </summary>
26 /// <param name="object"><para>The object.</para><para>Объект.</para></param>
27 /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делегат
    ↳ <see cref="Action{T}"/>.</para></param>
28 [MethodImpl(MethodImplOptions.AggressiveInlining)]
29 public Disposable(T @object, Action<T> action)
30 {
31     Object = @object;

```

```

32     OnDispose += (manual, wasDisposed) =>
33     {
34         if (!wasDisposed)
35         {
36             action(Object);
37         }
38     };
39 }
40
41 /// <summary>
42 /// <para>Initializes a new instance of the <see cref="Disposable{T}" /> object.</para>
43 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}" />.</para>
44 /// </summary>
45 /// <param name="object"><para>The object.</para><para>Объект.</para></param>
46 /// <param name="action"><para>The <see cref="Action" /> delegate.</para><para>Делегат
47   ↳ <see cref="Action" />.</para></param>
48 [MethodImpl(MethodImplOptions.AggressiveInlining)]
49 public Disposable(T @object, Action action) : base(action) => 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 /// <param name="disposal"><para>The <see cref="Disposal" />
57   ↳ delegate.</para><para>Делегат <see cref="Disposal" />.</para></param>
58 [MethodImpl(MethodImplOptions.AggressiveInlining)]
59 public Disposable(T @object, Disposal disposal) : base(disposal) => Object = @object;
60
61 /// <summary>
62 /// <para>Initializes a new instance of the <see cref="Disposable{T}" /> object.</para>
63 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}" />.</para>
64 /// </summary>
65 /// <param name="object"><para>The object.</para><para>Объект.</para></param>
66 [MethodImpl(MethodImplOptions.AggressiveInlining)]
67 public Disposable(T @object) => Object = @object;
68
69 /// <summary>
70 /// <para>Creates a new <see cref="Disposable{T}" /> object initialized with <see
71   ↳ cref="ValueTuple{T, TDisposal}.Item1" /> as <see cref="Disposable{T}.Object" /> and
72   ↳ <see cref="ValueTuple{T, TAction}.Item2" /> as delegate <see
73   ↳ cref="Action{T}" />.</para>
74 /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
75   ↳ <see cref="ValueTuple{T, TDisposal}.Item1" /> как <see cref="Disposable{T}.Object" />
76   ↳ и <see cref="ValueTuple{T, TAction}.Item2" /> как делегат <see
77   ↳ cref="Action{T}" />.</para>
78 /// </summary>
79 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
80 [MethodImpl(MethodImplOptions.AggressiveInlining)]
81 public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
82   ↳ Disposable<T>(tuple.Item1, tuple.Item2);
83
84 /// <summary>
85 /// <para>Creates a new <see cref="Disposable{T}" /> object initialized with <see
86   ↳ cref="ValueTuple{T, TDisposal}.Item1" /> as <see cref="Disposable{T}.Object" /> and
87   ↳ <see cref="ValueTuple{T, TDisposal}.Item2" /> as delegate <see
88   ↳ cref="Action{T}" />.</para>
89 /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
90   ↳ <see cref="ValueTuple{T, TDisposal}.Item1" /> как <see cref="Disposable{T}.Object" />
91   ↳ и <see cref="ValueTuple{T, TDisposal}.Item2" /> как делегат <see
92   ↳ cref="Action{T}" />.</para>
93 /// </summary>
94 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
95 [MethodImpl(MethodImplOptions.AggressiveInlining)]
96 public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
97   ↳ Disposable<T>(tuple.Item1, tuple.Item2);
98
99 /// <summary>
100 /// <para>Creates a new <see cref="Disposable{T}" /> object initialized with <see
101   ↳ cref="ValueTuple{T, TDisposal}.Item1" /> as <see cref="Disposable{T}.Object" /> and
102   ↳ <see cref="ValueTuple{T, TDisposal}.Item2" /> as delegate <see
103   ↳ cref="Disposal" />.</para>
104 /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
105   ↳ <see cref="ValueTuple{T, TDisposal}.Item1" /> как <see cref="Disposable{T}.Object" />
106   ↳ и <see cref="ValueTuple{T, TDisposal}.Item2" /> как делегат <see
107   ↳ cref="Disposal" />.</para>
108 /// </summary>
109 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
110 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

89     public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
90         ↳ Disposable<T>(tuple.Item1, tuple.Item2);
91
92     /// <summary>
93     /// <para>Creates a new <see cref="Disposable{T}"> object initialized with specified
94     ↳ object as <see cref="Disposable{T}.Object"/>.</para>
95     /// <para>Создает новый объект <see cref="Disposable{T}">, инициализированную с помощью
96     ↳ указанного объекта как <see cref="Disposable{T}.Object"/>.</para>
97     /// </summary>
98     /// <param name="object"><para>The object.</para><para>Объект.</para></param>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
101
102    /// <summary>
103    /// <para>Creates a new copy of the primary object (<see
104    ↳ cref="Disposable{T}.Object"/>).</para>
105    /// <para>Создаёт новую копию основного объекта (<see
106    ↳ cref="Disposable{T}.Object"/>).</para>
107    /// </summary>
108    /// <param name="disposableContainer"><para>The disposable
109    ↳ container.</para><para>Высвобождаемый контейнер.</para></param>
110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
111    public static implicit operator T(Disposable<T> disposableContainer) =>
112        ↳ disposableContainer.Object;
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
120    ↳ developer's code) or was executed automatically without an explicit indication from
121    ↳ a developer.</para>
122    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
123    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
124    ↳ разработчика.</para>
125    /// </param>
126    /// <param name="wasDisposed">
127    /// <para>A value that determines whether the object was released before calling this
128    ↳ method.</para>
129    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
130    /// </param>
131    [MethodImpl(MethodImplOptions.AggressiveInlining)]
132    protected override void Dispose(bool manual, bool wasDisposed)
133    {
134        base.Dispose(manual, wasDisposed);
135        Object.TryDispose();
136    }
137 }
138 }

```

1.5 ./csharp/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
6      ↳ ресурсов.</para>
7      /// </summary>
8      /// <param name="manual">
9      /// <para>A value that determines whether the disposal was triggered manually (by the
10     ↳ developer's code) or was executed automatically without an explicit indication from a
11     ↳ developer.</para>
12     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом разработчика)
13     ↳ или же выполнилось автоматически без явного указания со стороны разработчика.</para>
14     /// </param>
15     /// <param name="wasDisposed">
16     /// <para>A value that determines whether the object was released before calling this
17     ↳ method.</para>
18     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
19     /// </param>
20     public delegate void Disposal(bool manual, bool wasDisposed);
21 }

```

1.6 ./csharp/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
13     ↪ cref="Platform.Exceptions.ExtensionRoots.EnsureAlwaysExtensionRoot"/> and <see
14     ↪ cref="Platform.Exceptions.ExtensionRoots.EnsureOnDebugExtensionRoot"/> objects.</para>
15     /// <para>Предоставляет набор методов расширения для объектов <see
16     ↪ cref="Platform.Exceptions.ExtensionRoots.EnsureAlwaysExtensionRoot"/> и <see
17     ↪ cref="Platform.Exceptions.ExtensionRoots.EnsureOnDebugExtensionRoot"/>.</para>
18     /// </summary>
19     public static class EnsureExtensions
20     {
21         #region Always
22
23         /// <summary>
24         /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
25         ↪ has not been released. This check is performed regardless of the build
26         ↪ configuration.</para>
27         /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
28         ↪ был высвобожден. Эта проверка выполняется независимо от конфигурации
29         ↪ сборки.</para>
30         /// </summary>
31         /// <param name="root"><para>The extension root to which this method is
32         ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
33         /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
34         ↪ interface.</para><para>Объект, реализующий интерфейс <see
35         ↪ cref="IDisposable"/></para></param>
36         /// <param name="objectName"><para>The name of object.</para><para>Имя
37         ↪ объекта.</para></param>
38         /// <param name="message"><para>The message of the thrown
39         ↪ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
42         ↪ disposable, string objectName, string message)
43         {
44             if (disposable.IsDisposed)
45             {
46                 throw new ObjectDisposedException(objectName, message);
47             }
48         }
49
50         /// <summary>
51         /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
52         ↪ has not been released. This check is performed regardless of the build
53         ↪ configuration.</para>
54         /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
55         ↪ был высвобожден. Эта проверка выполняется независимо от конфигурации
56         ↪ сборки.</para>
57         /// </summary>
58         /// <param name="root"><para>The extension root to which this method is
59         ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
60         /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
61         ↪ interface.</para><para>Объект, реализующий интерфейс <see
62         ↪ cref="IDisposable"/></para></param>
63         /// <param name="objectName"><para>The name of object.</para><para>Имя
64         ↪ объекта.</para></param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
67         ↪ disposable, string objectName) => NotDisposed(root, disposable, objectName, null);
68
69         /// <summary>
70         /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
71         ↪ has not been released. This check is performed regardless of the build
72         ↪ configuration.</para>
73         /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
74         ↪ был высвобожден. Эта проверка выполняется независимо от конфигурации
75         ↪ сборки.</para>
76         /// </summary>
77         /// <param name="root"><para>The extension root to which this method is
78         ↪ 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 освобожден. 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 освобожден. 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 освобожден. 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 ./csharp/Platform.Disposables/GenericObjectExtensions.cs

```

1  using System;
2  using System.Runtime.CompilerServices;

```

```

3 using Platform.Exceptions;
4
5 namespace Platform.Disposables
6 {
7     /// <summary>
8     /// <para>Provides a set of static methods that help dispose a generic objects.</para>
9     /// <para>Предоставляет набор статических методов которые помогают высвободить универсальные
10    /// </summary>
11    static public class GenericObjectExtensions
12    {
13        /// <summary>
14        /// <para>Attempts to dispose the specified object.</para>
15        /// <para>Выполняет попытку высвободить указанный объект.</para>
16        /// </summary>
17        /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
18        /// <para>объекта.</para></typeparam>
19        /// <param name="object"><para>The object to dispose.</para><para>Объект, который
20        /// <para>необходимо высвободить.</para></param>
21        /// <returns><para>A value that determines whether the attempt to release the specified
22        /// <para>object was successful.</para><para>Значение, определяющие удачно ли была выполнена
23        /// <para>попытка высвободить указанный объект.</para></returns>
24        public static bool TryDispose<T>(this T @object)
25        {
26            try
27            {
28                if (@object is DisposableBase disposableBase)
29                {
30                    disposableBase.DisposeIfNotDisposed();
31                }
32                else if (@object is System.IDisposable disposable)
33                {
34                    disposable.Dispose();
35                }
36                return true;
37            }
38            catch (Exception exception)
39            {
40                exception.Ignore();
41            }
42            return false;
43        }
44
45        /// <summary>
46        /// <para>Attempts to dispose the specified object.</para>
47        /// <para>Выполняет попытку высвободить указанный объект.</para>
48        /// </summary>
49        /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
50        /// <para>объекта.</para></typeparam>
51        /// <param name="object"><para>The object to dispose.</para><para>Объект, который
52        /// <para>необходимо высвободить.</para></param>
53        [MethodImpl(MethodImplOptions.AggressiveInlining)]
54        public static void DisposeIfPossible<T>(this T @object) => TryDispose(@object);
55    }
56 }

```

1.8 ./csharp/Platform.Disposables/IDisposable.cs

```

1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Disposables
4 {
5     /// <summary>
6     /// <para>Представляет расширенный интерфейс <see cref="System.IDisposable"/>.</para>
7     /// <para>Represents an extended <see cref="System.IDisposable"/> interface.</para>
8     /// </summary>
9     public interface IDisposable : System.IDisposable
10    {
11        /// <summary>
12        /// <para>Gets a value indicating whether the object was disposed.</para>
13        /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
14        /// </summary>
15        bool IsDisposed
16        {
17            [MethodImpl(MethodImplOptions.AggressiveInlining)]
18            get;
19        }
20
21        /// <summary>

```

```

22     /// <para>Performs application-defined tasks associated with freeing, releasing, or
    ↪ resetting unmanaged resources without throwing any exceptions.</para>
23     /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    ↪ высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
24     /// </summary>
25     /// <remarks>
26     /// <para>Should be called only from classes destructors, or in case exceptions should
    ↪ be not thrown.</para>
27     /// <para>Должен вызываться только из деструкторов классов, или в случае, если
    ↪ исключения выбрасывать нельзя.</para>
28     /// </remarks>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     void Destruct();
31 }
32 }

```

1.9 ./csharp/Platform.Disposables/IDisposableExtensions.cs

```

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

```

1.10 ./csharp/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     /// <summary>
12     /// <para>
13     /// Represents the disposable tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class DisposableTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that disposal order test.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         [Fact]
26         public static void DisposalOrderTest()
27         {
28             var logPath = Path.GetTempFileName();
29             using (var process = Process.Start(CreateProcessStartInfo(logPath,
    ↪ waitForCancellation: false)))
30             {

```

```

31         process.WaitForExit();
32     }
33     var result = File.ReadAllText(logPath);
34     Assert.Equal("21", result);
35     File.Delete(logPath);
36 }
37
38 /// <summary>
39 /// <para>
40 /// Tests that disposal at process kill test.
41 /// </para>
42 /// <para></para>
43 /// </summary>
44 [Fact]
45 public static void DisposalAtProcessKillTest()
46 {
47     var logPath = Path.GetTempFileName();
48     using (var process = Process.Start(CreateProcessStartInfo(logPath,
49         ↪ waitForCancellation: true)))
50     {
51         Thread.Sleep(1000);
52         process.Kill();
53     }
54     var result = File.ReadAllText(logPath);
55     Assert.Equal("", result); // Currently, process termination will not release
56     ↪ resources
57     File.Delete(logPath);
58 }
59
60 /// <summary>
61 /// <para>
62 /// Creates the process start info using the specified log path.
63 /// </para>
64 /// <para></para>
65 /// </summary>
66 /// <param name="logPath">
67 /// <para>The log path.</para>
68 /// <para></para>
69 /// </param>
70 /// <param name="waitForCancellation">
71 /// <para>The wait for cancellation.</para>
72 /// <para></para>
73 /// </param>
74 /// <returns>
75 /// <para>The process start info</para>
76 /// <para></para>
77 /// </returns>
78 private static ProcessStartInfo CreateProcessStartInfo(string logPath, bool
79     ↪ waitForCancellation)
80 {
81     var projectPath = GetDisposalObjectTestProjectFilePath();
82     return new ProcessStartInfo
83     {
84         FileName = "dotnet",
85         Arguments = $"run -p \"{projectPath}\" -f net5 \"{logPath}\"
86         ↪ {waitForCancellation.ToString()}",
87         UseShellExecute = false,
88         CreateNoWindow = true
89     };
90 }
91
92 /// <summary>
93 /// <para>
94 /// Gets the disposal object test project file path.
95 /// </para>
96 /// <para></para>
97 /// </summary>
98 /// <returns>
99 /// <para>The path.</para>
100 /// <para></para>
101 /// </returns>
102 private static string GetDisposalObjectTestProjectFilePath()
103 {
104     const string currentProjectName = nameof(Platform) + "." + nameof(Disposables) + "."
105     ↪ + nameof(Tests);
106     const string disposalOrderTestProjectName = currentProjectName + "." +
107     ↪ nameof(DisposalOrderTest);
108     var currentDirectory = Environment.CurrentDirectory;

```



```

103     var pathParts = currentDirectory.Split(Path.DirectorySeparatorChar);
104     var newPathParts = new List<string>();
105     for (var i = 0; i < pathParts.Length; i++)
106     {
107         if (string.Equals(pathParts[i], currentProjectName))
108         {
109             newPathParts.Add(disposalOrderTestProjectName);
110             break;
111         }
112         else
113         {
114             newPathParts.Add(pathParts[i]);
115         }
116     }
117     pathParts = newPathParts.ToArray();
118     #if NET472
119     var directory = string.Join(Path.DirectorySeparatorChar.ToString(),
120         ↪ pathParts.ToArray());
121     #else
122     var directory = Path.Combine(pathParts);
123     #endif
124     var path = Path.Combine(directory, $"{disposalOrderTestProjectName}.csproj");
125     if (!Path.IsPathRooted(path))
126     {
127         path = $"{Path.DirectorySeparatorChar}{path}";
128     }
129     return path;
130 }
131 }

```

1.11 ./csharp/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         /// <summary>
14         /// <para>
15         /// Tests that using supports null test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void UsingSupportsNullTest()
21         {
22             using var disposable = null as IDisposable;
23             Assert.True(disposable == null);
24         }
25     }
26 }

```

Index

- ./csharp/Platform.Disposables.Tests/DisposableTests.cs, 15
- ./csharp/Platform.Disposables.Tests/SystemTests.cs, 17
- ./csharp/Platform.Disposables/Disposable.cs, 1
- ./csharp/Platform.Disposables/DisposableBase.cs, 3
- ./csharp/Platform.Disposables/Disposable[TPrimary, TAuxiliary].cs, 6
- ./csharp/Platform.Disposables/Disposable[T].cs, 9
- ./csharp/Platform.Disposables/Disposal.cs, 11
- ./csharp/Platform.Disposables/EnsureExtensions.cs, 11
- ./csharp/Platform.Disposables/GenericObjectExtensions.cs, 13
- ./csharp/Platform.Disposables/IDisposable.cs, 14
- ./csharp/Platform.Disposables/IDisposableExtensions.cs, 15