

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        private static readonly Disposal _emptyDelegate = (manual, wasDisposed) => { };
15
16        /// <summary>
17        /// <para>Occurs when the object is being disposed.</para>
18        /// <para>Возникает, когда объект высвобождается.</para>
19        /// </summary>
20        public event Disposal OnDispose;
21
22        /// <summary>
23        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
24        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
25        /// </summary>
26        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
27        ///     ↳ <see cref="Action"/>.</para></param>
28        [MethodImpl(MethodImplOptions.AggressiveInlining)]
29        public Disposable(Action action)
30        {
31            OnDispose = (manual, wasDisposed) =>
32            {
33                if (!wasDisposed)
34                {
35                    action();
36                }
37            };
38        }
39
40        /// <summary>
41        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
42        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
43        /// </summary>
44        /// <param name="disposal"><para>The <see cref="Disposal"/>
45        ///     ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
46        [MethodImpl(MethodImplOptions.AggressiveInlining)]
47        public Disposable(Disposal disposal) => OnDispose = disposal;
48
49        /// <summary>
50        /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
51        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
52        /// </summary>
53        [MethodImpl(MethodImplOptions.AggressiveInlining)]
54        public Disposable() => OnDispose = _emptyDelegate;
55
56        /// <summary>
57        /// <para>Creates a new <see cref="Disposable"/> object initialized with specified
58        ///     ↳ delegate <see cref="Action"/>.</para>
59        /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
60        ///     ↳ указанного делегата <see cref="Action"/>.</para>
61        /// </summary>
62        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
63        ///     ↳ <see cref="Action"/>.</para></param>
64        [MethodImpl(MethodImplOptions.AggressiveInlining)]
65        public static implicit operator Disposable(Action action) => new Disposable(action);
66
67        /// <summary>
68        /// <para>Creates a new <see cref="Disposable"/> object initialized with specified
69        ///     ↳ delegate <see cref="Disposal"/>.</para>
70        /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
71        ///     ↳ указанного делегата <see cref="Disposal"/>.</para>
72        /// </summary>
73        /// <param name="disposal"><para>The <see cref="Disposal"/>
74        ///     ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
75        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```

66     public static implicit operator Disposable(Disposal disposal) => new
        ↳ Disposable(disposal);
67
68     /// <summary>
69     /// <para>Disposes unmanaged resources.</para>
70     /// <para>Высвобождает неуправляемые ресурсы.</para>
71     /// </summary>
72     /// <param name="manual">
73     /// <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>
74     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
        ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
        ↳ разработчика.</para>
75     /// </param>
76     /// <param name="wasDisposed">
77     /// <para>A value that determines whether the object was released before calling this
        ↳ method.</para>
78     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
79     /// </param>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     protected override void Dispose(bool manual, bool wasDisposed) =>
        ↳ RaiseOnDisposeEvent(manual, wasDisposed);
82
83     /// <summary>
84     /// <para>Raises an unmanaged resource dispose event.</para>
85     /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
86     /// </summary>
87     /// <param name="manual">
88     /// <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>
89     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
        ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
        ↳ разработчика.</para>
90     /// </param>
91     /// <param name="wasDisposed">
92     /// <para>A value that determines whether the object was released before calling this
        ↳ method.</para>
93     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
        ↳ wasDisposed);
97
98     /// <summary>
99     /// <para>Attempts to dispose the specified object, as well as set the value of the
        ↳ variable containing this object to the default value.</para>
100    /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
        ↳ переменной содержащей этот объект в значение по умолчанию.</para>
101    /// </summary>
102    /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
        ↳ объекта.</para></typeparam>
103    /// <param name="object"><para>The object to dispose.</para><para>Объект, который
        ↳ необходимо высвободить.</para></param>
104    /// <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>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    public static bool TryDisposeAndResetToDefault<T>(ref T @object)
107    {
108        var result = @object.TryDispose();
109        if (result)
110        {
111            @object = default;
112        }
113        return result;
114    }
115 }
116 }

```

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>Provides a base implementation for IDisposable interface with the basic logic
11    /// <para>↳ necessary to increase the likelihood of correct unmanaged resources release.</para>
12    /// <para>Предоставляет базовую реализацию для интерфейса IDisposable с основной логикой
13    /// <para>↳ необходимой для повышения вероятности корректного высвобождения неуправляемых
14    /// <para>↳ ресурсов.</para>
15    /// </summary>
16    public abstract class DisposableBase : IDisposable
17    {
18        private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
19        private static readonly ConcurrentStack<WeakReference<DisposableBase>>
20        _disposablesWeekReferencesStack = new
21        ConcurrentStack<WeakReference<DisposableBase>>();
22
23        private volatile int _disposed;
24
25        /// <summary>
26        /// <para>Gets a value indicating whether the object was disposed.</para>
27        /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
28        /// </summary>
29        public bool IsDisposed
30        {
31            [MethodImpl(MethodImplOptions.AggressiveInlining)]
32            get => _disposed > 0;
33        }
34
35        /// <summary>
36        /// <para>Gets the name of an object or a unique string describing this object.</para>
37        /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
38        /// </summary>
39        protected virtual string ObjectName
40        {
41            [MethodImpl(MethodImplOptions.AggressiveInlining)]
42            get => GetType().Name;
43        }
44
45        /// <summary>
46        /// <para>Gets a value indicating whether multiple attempts to dispose this object are
47        /// <para>↳ allowed.</para>
48        /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток
49        /// <para>↳ высвободить этот объект.</para>
50        /// </summary>
51        protected virtual bool AllowMultipleDisposeAttempts
52        {
53            [MethodImpl(MethodImplOptions.AggressiveInlining)]
54            get => false;
55        }
56
57        /// <summary>
58        /// <para>Gets a value indicating whether it is allowed to call this object disposal
59        /// <para>↳ multiple times.</para>
60        /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
61        /// <para>↳ высвобождение этого объекта.</para>
62        /// </summary>
63        protected virtual bool AllowMultipleDisposeCalls
64        {
65            [MethodImpl(MethodImplOptions.AggressiveInlining)]
66            get => false;
67        }
68
69        [MethodImpl(MethodImplOptions.AggressiveInlining)]
70        static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
71
72        /// <summary>
73        /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
74        /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
75        /// </summary>
76        [MethodImpl(MethodImplOptions.AggressiveInlining)]
77        protected DisposableBase()
78        {
79            _disposed = 0;
80            _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
81        }
82
83        /// <summary>

```

```

75     /// <para>Performs any necessary final clean-up when a class instance is being collected
76     → by the garbage collector.</para>
77     /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
78     → собирает экземпляр класса.</para>
79     /// </summary>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     ~DisposableBase() => Destruct();
82
83     /// <summary>
84     /// <para>Disposes unmanaged resources.</para>
85     /// <para>Высвобождает неуправляемые ресурсы.</para>
86     /// </summary>
87     /// <param name="manual">
88     /// <para>A value that determines whether the disposal was triggered manually (by the
89     → developer's code) or was executed automatically without an explicit indication from
90     → the developer.</para>
91     /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
92     → разработчика) или же выполнилось автоматически без явного указания со стороны
93     → разработчика.</para>
94     /// </param>
95     /// <param name="wasDisposed">
96     /// <para>A value that determines whether the object was released before calling this
97     → method.</para>
98     /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
99     /// </param>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    protected abstract void Dispose(bool manual, bool wasDisposed);
102
103    /// <summary>
104    /// <para>Performs application-defined tasks associated with freeing, releasing, or
105    → resetting unmanaged resources.</para>
106    /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
107    → высвобождением или сбросом неуправляемых ресурсов.</para>
108    /// </summary>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    public void Dispose()
111    {
112        Dispose(true);
113        GC.SuppressFinalize(this);
114    }
115
116    /// <summary>
117    /// <para>Performs application-defined tasks associated with freeing, releasing, or
118    → resetting unmanaged resources without throwing any exceptions.</para>
119    /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
120    → высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
121    /// </summary>
122    /// <remarks>
123    /// <para>Should be called only from classes destructors, or in case exceptions should
124    → be not thrown.</para>
125    /// <para>Должен вызываться только из деструкторов классов, или в случае, если
126    → исключения выбрасывать нельзя.</para>
127    /// </remarks>
128    [MethodImpl(MethodImplOptions.AggressiveInlining)]
129    public void Destruct()
130    {
131        try
132        {
133            if (!IsDisposed)
134            {
135                Dispose(false);
136            }
137        }
138        catch (Exception exception)
139        {
140            exception.Ignore();
141        }
142    }
143
144    /// <summary>
145    /// <para>Disposes unmanaged resources.</para>
146    /// <para>Высвобождает неуправляемые ресурсы.</para>
147    /// </summary>
148    /// <param name="manual">
149    /// <para>A value that determines whether the disposal was triggered manually (by the
150    → developer's code) or was executed automatically without an explicit indication from
151    → the developer.</para>

```

```

137  /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↳ разработчика.</para>
138  /// </param>
139  protected virtual void Dispose(bool manual)
140  {
141      var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
142      var wasDisposed = originalDisposedValue > 0;
143      if (wasDisposed && !AllowMultipleDisposeCalls && manual)
144      {
145          Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not
            ↳ allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
146      }
147      if (AllowMultipleDisposeAttempts || !wasDisposed)
148      {
149          Dispose(manual, wasDisposed);
150      }
151  }
152
153  private static void OnProcessExit(object sender, EventArgs e)
154  {
155      while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
            ↳ weakReference))
156      {
157          if (weakReference.TryGetTarget(out DisposableBase disposable))
158          {
159              GC.SuppressFinalize(disposable);
160              disposable.Destruct();
161          }
162      }
163      UnsubscribeFromProcessExitedEventIfPossible();
164  }
165
166  private static void UnsubscribeFromProcessExitedEventIfPossible()
167  {
168      try
169      {
170          if (_currentDomain != null)
171          {
172              _currentDomain.ProcessExit -= OnProcessExit;
173          }
174          else
175          {
176              AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
177          }
178      }
179      catch (Exception exception)
180      {
181          exception.Ignore();
182      }
183  }
184  }
185  }

```

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
            ↳ container itself is disposed.</para>
8      /// <para>Представляет высвобождаемый контейнер, который высвобождает два содержащихся в
            ↳ нём объектов при высвобождении самого контейнера.</para>
9      /// </summary>
10     /// <typeparam name="TPrimary"><para>The primary object type.</para><para>Тип основного
            ↳ объекта.</para></typeparam>
11     /// <typeparam name="TAuxiliary"><para>The auxiliary object type.</para><para>Тип
            ↳ вспомогательного объекта.</para></typeparam>
12     public class Disposable<TPrimary, TAuxiliary> : Disposable<TPrimary>
13     {
14         /// <summary>
15         /// <para>Gets the auxiliary object.</para>
16         /// <para>Возвращает вспомогательный объект.</para>
17         /// </summary>
18         public TAuxiliary AuxiliaryObject
19         {

```

```

20     [MethodImpl(MethodImplOptions.AggressiveInlining)]
21     get;
22 }
23
24 /// <summary>
25 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para> object.</para>
26 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para>.</para>
27 /// </summary>
28 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
29 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
30 /// <param name="action"><para>The <see cref="Action{TPrimary, TAuxiliary}</para>
    ↳ delegate.</para><para>Делегат <see cref="Action{TPrimary,
    ↳ TAuxiliary}</para>.</para></param>
31 [MethodImpl(MethodImplOptions.AggressiveInlining)]
32 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPrimary,
    ↳ TAuxiliary> action)
33     : base(@object)
34 {
35     AuxiliaryObject = auxiliaryObject;
36     OnDispose += (manual, wasDisposed) =>
37     {
38         if (!wasDisposed)
39         {
40             action(Object, AuxiliaryObject);
41         }
42     };
43 }
44
45 /// <summary>
46 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para> object.</para>
47 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para>.</para>
48 /// </summary>
49 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
50 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
51 /// <param name="action"><para>The <see cref="Action"> delegate.</para><para>Делегат
    ↳ <see cref="Action">.</para></param>
52 [MethodImpl(MethodImplOptions.AggressiveInlining)]
53 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
    ↳ base(@object, action) => AuxiliaryObject = auxiliaryObject;
54
55 /// <summary>
56 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para> object.</para>
57 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para>.</para>
58 /// </summary>
59 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
60 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
61 /// <param name="disposal"><para>The <see cref="Disposal">
    ↳ delegate.</para><para>Делегат <see cref="Disposal">.</para></param>
62 [MethodImpl(MethodImplOptions.AggressiveInlining)]
63 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
    ↳ base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
64
65 /// <summary>
66 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para> object.</para>
67 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}</para>.</para>
68 /// </summary>
69 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
70 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
71 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

72 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
73     ↪ AuxiliaryObject = auxiliaryObject;
74
75 /// <summary>
76 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
77     ↪ TAuxiliary}" /> object.</para>
78 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
79     ↪ TAuxiliary}" />.</para>
80 /// </summary>
81 /// <param name="object"><para>The primary object.</para><para>Основной
82     ↪ объект.</para></param>
83 [MethodImpl(MethodImplOptions.AggressiveInlining)]
84 public Disposable(TPrimary @object) : base(@object) { }
85
86 /// <summary>
87 /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}" /> object
88     ↪ initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1" /> as
89     ↪ <see cref="Disposable{TPrimary}.Object" />, <see cref="ValueTuple{TPrimary,
90     ↪ TAuxiliary, TAction}.Item2" /> as <see cref="Disposable{TPrimary,
91     ↪ TAuxiliary}.AuxiliaryObject" /> and <see cref="ValueTuple{TPrimary, TAuxiliary,
92     ↪ TAction}.Item3" /> as delegate <see cref="Action{TPrimary, TAuxiliary}" />.</para>
93 /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}" />,
94     ↪ инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
95     ↪ TAction}.Item1" /> как <see cref="Disposable{TPrimary}.Object" />, <see
96     ↪ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2" /> как <see
97     ↪ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject" /> и <see
98     ↪ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3" /> как делегат <see
99     ↪ cref="Action{TPrimary, TAuxiliary}" />.</para>
100 /// </summary>
101 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
102 [MethodImpl(MethodImplOptions.AggressiveInlining)]
103 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↪ TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
    ↪ TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);

```

```

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>Создаёт новый объект <see cref="Disposable{TPrimary, TAuxiliary}" />,
    ↳ инициализированный 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
    ↳ the 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
8     /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
9     /// <para>объект при высвобождении самого контейнера.</para>
10    /// </summary>
11    public class Disposable<T> : Disposable
12    {
13        /// <summary>
14        /// <para>Gets the object.</para>
15        /// <para>Возвращает объект.</para>
16        /// </summary>
17        public T Object
18        {
19            [MethodImpl(MethodImplOptions.AggressiveInlining)]
20            get;
21        }
22
23        /// <summary>
24        /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
25        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
26        /// </summary>
27        /// <param name="object"><para>The object.</para><para>Объект.</para></param>
28        /// <param name="action"><para>The <see cref="Action{T}"> delegate.</para><para>Делегат
29        /// <para><see cref="Action{T}">.</para></param>
30        [MethodImpl(MethodImplOptions.AggressiveInlining)]
31        public Disposable(T @object, Action<T> action)
32        {
33            Object = @object;
34            OnDispose += (manual, wasDisposed) =>
35            {
36                if (!wasDisposed)
37                {
38                    action(Object);
39                }
40            };
41        }
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="action"><para>The <see cref="Action"> delegate.</para><para>Делегат
49        /// <para><see cref="Action">.</para></param>
50        [MethodImpl(MethodImplOptions.AggressiveInlining)]
51        public Disposable(T @object, Action action) : base(action) => Object = @object;
52
53        /// <summary>
54        /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
55        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
56        /// </summary>
57        /// <param name="object"><para>The object.</para><para>Объект.</para></param>
58        /// <param name="disposal"><para>The <see cref="Disposal">
59        /// <para>delegate.</para><para>Делегат <see cref="Disposal">.</para></param>
60        [MethodImpl(MethodImplOptions.AggressiveInlining)]
61        public Disposable(T @object, Disposal disposal) : base(disposal) => Object = @object;
62
63        /// <summary>
64        /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
65        /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
66        /// </summary>
67        /// <param name="object"><para>The object.</para><para>Объект.</para></param>
68        [MethodImpl(MethodImplOptions.AggressiveInlining)]
69        public Disposable(T @object) => Object = @object;
70
71        /// <summary>
72        /// <para>Creates a new <see cref="Disposable{T}"> object initialized with <see
73        /// <para>cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
74        /// <para><see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see
75        /// <para>cref="Action{T}">.</para>
76        /// <para>Создает новый объект <see cref="Disposable{T}">, инициализированную с помощью
77        /// <para><see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
78        /// <para>и <see cref="ValueTuple{T, TAction}.Item2"/> как делегат <see
79        /// <para>cref="Action{T}">.</para>

```

```

70     /// </summary>
71     /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
74     ↪ Disposable<T>(tuple.Item1, tuple.Item2);
75
76     /// <summary>
77     /// <para>Creates a new <see cref="Disposable{T}"> object initialized with <see
78     ↪ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
79     ↪ <see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see cref="Action"/>.</para>
80     /// <para>Создаёт новый объект <see cref="Disposable{T}">, инициализированную с помощью
81     ↪ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
82     ↪ и <see cref="ValueTuple{T, TAction}.Item2"/> как делегат <see cref="Action"/>.</para>
83     /// </summary>
84     /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
87     ↪ Disposable<T>(tuple.Item1, tuple.Item2);
88
89     /// <summary>
90     /// <para>Creates a new <see cref="Disposable{T}"> object initialized with <see
91     ↪ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
92     ↪ <see cref="ValueTuple{T, TDisposal}.Item2"/> as delegate <see
93     ↪ cref="Disposal"/>.</para>
94     /// <para>Создаёт новый объект <see cref="Disposable{T}">, инициализированную с помощью
95     ↪ <see cref="ValueTuple{T, TDisposal}.Item1"/> как <see cref="Disposable{T}.Object"/>
96     ↪ и <see cref="ValueTuple{T, TDisposal}.Item2"/> как делегат <see
97     ↪ cref="Disposal"/>.</para>
98     /// </summary>
99     /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
102    ↪ Disposable<T>(tuple.Item1, tuple.Item2);
103
104    /// <summary>
105    /// <para>Creates a new <see cref="Disposable{T}"> object initialized with specified
106    ↪ object as <see cref="Disposable{T}.Object"/>.</para>
107    /// <para>Создаёт новый объект <see cref="Disposable{T}">, инициализированную с помощью
108    ↪ указанного объекта как <see cref="Disposable{T}.Object"/>.</para>
109    /// </summary>
110    /// <param name="object"><para>The object.</para><para>Объект.</para></param>
111    [MethodImpl(MethodImplOptions.AggressiveInlining)]
112    public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
113
114    /// <summary>
115    /// <para>Creates a new copy of the primary object (<see
116    ↪ cref="Disposable{T}.Object"/>).</para>
117    /// <para>Создаёт новую копию основного объекта (<see
118    ↪ cref="Disposable{T}.Object"/>).</para>
119    /// </summary>
120    /// <param name="disposableContainer"><para>The disposable
121    ↪ container.</para><para>Высвобождаемый контейнер.</para></param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    public static implicit operator T(Disposable<T> disposableContainer) =>
124    ↪ disposableContainer.Object;
125
126    /// <summary>
127    /// <para>Disposes unmanaged resources.</para>
128    /// <para>Высвобождает неуправляемые ресурсы.</para>
129    /// </summary>
130    /// <param name="manual">
131    /// <para>A value that determines whether the disposal was triggered manually (by the
132    ↪ developer's code) or was executed automatically without an explicit indication from
133    ↪ the developer.</para>
134    /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
135    ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
136    ↪ разработчика.</para>
137    /// </param>
138    /// <param name="wasDisposed">
139    /// <para>A value that determines whether the object was released before calling this
140    ↪ method.</para>
141    /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
142    /// </param>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void Dispose(bool manual, bool wasDisposed)
145    {

```

```

122         base.Dispose(manual, wasDisposed);
123         Object.TryDispose();
124     }
125 }
126 }

```

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 the
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 }

```

```

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 ./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
    → объекты.</para>
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
    → объекта.</para></typeparam>
18         /// <param name="object"><para>The object to dispose.</para><para>Объект, который
    → необходимо высвободить.</para></param>
19         /// <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>
20         public static bool TryDispose<T>(this T @object)
21         {
22             try
23             {
24                 if (@object is DisposableBase disposableBase)
25                 {
26                     disposableBase.DisposeIfNotDisposed();
27                 }
28                 else if (@object is System.IDisposable disposable)
29                 {
30                     disposable.Dispose();
31                 }
32                 return true;
33             }
34             catch (Exception exception)
35             {
36                 exception.Ignore();
37             }
38             return false;
39         }
40     }

```

```

41     /// <summary>
42     /// <para>Attempts to dispose the specified object.</para>
43     /// <para>Выполняет попытку высвободить указанный объект.</para>
44     /// </summary>
45     /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
    ↪ объекта.</para></typeparam>
46     /// <param name="object"><para>The object to dispose.</para><para>Объект, который
    ↪ необходимо высвободить.</para></param>
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public static void DisposeIfPossible<T>(this T @object) => TryDispose(@object);
49 }
50 }

```

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 }

```

```

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     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             }
74             else
75             {
76

```

```

68         newPathParts.Add(pathParts[i]);
69     }
70 }
71 pathParts = newPathParts.ToArray();
72 #if NET472
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 ./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         [Fact]
14         public static void UsingSupportsNullTest()
15         {
16             using var disposable = null as IDisposable;
17             Assert.True(disposable == null);
18         }
19     }
20 }

```


Index

- ./csharp/Platform.Disposables.Tests/DisposableTests.cs, 15
- ./csharp/Platform.Disposables.Tests/SystemTests.cs, 16
- ./csharp/Platform.Disposables/Disposable.cs, 1
- ./csharp/Platform.Disposables/DisposableBase.cs, 2
- ./csharp/Platform.Disposables/Disposable[TPrimary, TAuxiliary].cs, 5
- ./csharp/Platform.Disposables/Disposable[T].cs, 8
- ./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, 14