

# LinksPlatform's Platform.Disposables Class Library

./Platform.Disposables/DisposableBase.cs

```

1  using System;
2  using System.Collections.Concurrent;
3  using System.Threading;
4  using Platform.Exceptions;
5
6  namespace Platform.Disposables
7  {
8      /// <summary>
9      /// <para>Provides a base implementation for IDisposable interface with the basic logic
10     ↪ necessary to increase the likelihood of correct unmanaged resources release.</para>
11     /// <para>Предоставляет базовую реализацию для интерфейса IDisposable с основной логикой
12     ↪ необходимой для повышения вероятности корректного высвобождения неуправляемых
13     ↪ ресурсов.</para>
14     /// </summary>
15     public abstract class DisposableBase : IDisposable
16     {
17         private static readonly AppDomain _currentDomain = AppDomain.CurrentDomain;
18         private static readonly ConcurrentStack<WeakReference<DisposableBase>>
19             ↪ _disposablesWeekReferencesStack = new
20             ↪ ConcurrentStack<WeakReference<DisposableBase>>();
21
22         private volatile int _disposed;
23
24         /// <summary>
25         /// <para>Gets a value indicating whether the object was disposed.</para>
26         /// <para>Возвращает значение определяющее был ли высвобожден объект.</para>
27         /// </summary>
28         public bool IsDisposed => _disposed > 0;
29
30         /// <summary>
31         /// <para>Gets the name of an object or a unique string describing this object.</para>
32         /// <para>Возвращает имя объекта или уникальную строку описывающую этот объект.</para>
33         /// </summary>
34         protected virtual string ObjectName => GetType().Name;
35
36         /// <summary>
37         /// <para>Gets a value indicating whether multiple attempts to dispose this object are
38             ↪ allowed.</para>
39         /// <para>Возвращает значение определяющие разрешено ли выполнять несколько попыток
40             ↪ высвободить этот объект.</para>
41         /// </summary>
42         protected virtual bool AllowMultipleDisposeAttempts => false;
43
44         /// <summary>
45         /// <para>Gets a value indicating whether it is allowed to call this object disposal
46             ↪ multiple times.</para>
47         /// <para>Возвращает значение определяющие разрешено ли несколько раз вызывать
48             ↪ высвобождение этого объекта.</para>
49         /// </summary>
50         protected virtual bool AllowMultipleDisposeCalls => false;
51
52         static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
53
54         /// <summary>
55         /// <para>Initializes a new instance of the <see cref="DisposableBase"/> class.</para>
56         /// <para>Инициализирует новый экземпляр класса <see cref="DisposableBase"/>.</para>
57         /// </summary>
58         protected DisposableBase()
59         {
60             _disposed = 0;
61             _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
62         }
63
64         /// <summary>
65         /// <para>Performs any necessary final clean-up when a class instance is being collected
66             ↪ by the garbage collector.</para>
67         /// <para>Выполняет любую необходимую окончательную очистку, когда сборщик мусора
68             ↪ собирает экземпляр класса.</para>
69         /// </summary>
70         ~DisposableBase() => Destruct();
71
72         /// <summary>
73         /// <para>Disposes unmanaged resources.</para>
74         /// <para>Высвобождает неуправляемые ресурсы.</para>
75         /// </summary>
76         /// <param name="manual">

```

```

66  /// <para>A value that determines whether the disposal was triggered manually (by the
    → developer's code) or was executed automatically without an explicit indication from
    → the developer.</para>
67  /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    → разработчика) или же выполнилось автоматически без явного указания со стороны
    → разработчика.</para>
68  /// </param>
69  /// <param name="wasDisposed">
70  /// <para>A value that determines whether the object was released before calling this
    → method.</para>
71  /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
72  /// </param>
73  protected abstract void Dispose(bool manual, bool wasDisposed);
74
75  /// <summary>
76  /// <para>Performs application-defined tasks associated with freeing, releasing, or
    → resetting unmanaged resources.</para>
77  /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    → высвобождением или сбросом неуправляемых ресурсов.</para>
78  /// </summary>
79  public void Dispose()
80  {
81      GC.SuppressFinalize(this);
82      Dispose(true);
83  }
84
85  /// <summary>
86  /// <para>Performs application-defined tasks associated with freeing, releasing, or
    → resetting unmanaged resources without throwing any exceptions.</para>
87  /// <para>Выполняет определенные пользователем задачи, связанные с освобождением,
    → высвобождением или сбросом неуправляемых ресурсов без выбрасывания исключений.</para>
88  /// </summary>
89  /// <remarks>
90  /// <para>Should be called only from classes destructors, or in case exceptions should
    → be not thrown.</para>
91  /// <para>Должен вызываться только из деструкторов классов, или в случае, если
    → исключения выбрасывать нельзя.</para>
92  /// </remarks>
93  public void Destruct()
94  {
95      try
96      {
97          if (!IsDisposed)
98          {
99              Dispose(false);
100          }
101      }
102      catch (Exception exception)
103      {
104          exception.Ignore();
105      }
106  }
107
108  private void Dispose(bool manual)
109  {
110      var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
111      var wasDisposed = originalDisposedValue > 0;
112      if (wasDisposed && !AllowMultipleDisposeCalls && manual)
113      {
114          Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are not
            → allowed. Override AllowMultipleDisposeCalls property to modify behavior.");
115      }
116      if (AllowMultipleDisposeAttempts || !wasDisposed)
117      {
118          Dispose(manual, wasDisposed);
119      }
120  }
121
122  private static void OnProcessExit(object sender, EventArgs e)
123  {
124      while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
            → weakReference))
125      {
126          if (weakReference.TryGetTarget(out DisposableBase disposable))
127          {
128              GC.SuppressFinalize(disposable);
129              disposable.Destruct();

```

```

130     }
131 }
132 UnsubscribeFromProcessExitedEventIfPossible();
133 }
134
135 private static void UnsubscribeFromProcessExitedEventIfPossible()
136 {
137     try
138     {
139         if (_currentDomain != null)
140         {
141             _currentDomain.ProcessExit -= OnProcessExit;
142         }
143         else
144         {
145             AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
146         }
147     }
148     catch (Exception exception)
149     {
150         exception.Ignore();
151     }
152 }
153 }
154 }

```

./Platform.Disposables/Disposable.cs

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// <para>Represents disposable object that contains OnDispose event which is raised when
7      ///     the object itself is disposed.</para>
8      /// <para>Представляет высвобождаемый объект, который содержит событие OnDispose, которое
9      ///     возникает при высвобождении самого объекта.</para>
10     /// </summary>
11     public class Disposable : DisposableBase
12     {
13         private static readonly Disposal _emptyDelegate = (manual, wasDisposed) => { };
14
15         /// <summary>
16         /// <para>Occurs when the object is being disposed.</para>
17         /// <para>Возникает, когда объект высвобождается.</para>
18         /// </summary>
19         public event Disposal OnDispose;
20
21         /// <summary>
22         /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
23         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
24         /// </summary>
25         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
26         ///     <see cref="Action"/>.</para></param>
27         public Disposable(Action action)
28         {
29             OnDispose = (manual, wasDisposed) =>
30             {
31                 if (!wasDisposed)
32                 {
33                     action();
34                 }
35             };
36         }
37
38         /// <summary>
39         /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
40         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
41         /// </summary>
42         /// <param name="disposal"><para>The <see cref="Disposal"/>
43         ///     delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
44         public Disposable(Disposal disposal) => OnDispose = disposal;
45
46         /// <summary>
47         /// <para>Initializes a new instance of the <see cref="Disposable"/> object.</para>
48         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable"/>.</para>
49         /// </summary>
50         public Disposable() => OnDispose = _emptyDelegate;
51
52         /// <summary>

```

```

49  /// <para>Creates a new <see cref="Disposable"/> object initialized with specified
    ↳ delegate <see cref="Action"/>.</para>
50  /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
    ↳ указанного делегата <see cref="Action"/>.</para>
51  /// </summary>
52  /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
    ↳ <see cref="Action"/>.</para></param>
53  public static implicit operator Disposable(Action action) => new Disposable(action);
54
55  /// <summary>
56  /// <para>Creates a new <see cref="Disposable"/> object initialized with specified
    ↳ delegate <see cref="Disposal"/>.</para>
57  /// <para>Создает новый объект <see cref="Disposable"/>, инициализированную с помощью
    ↳ указанного делегата <see cref="Disposal"/>.</para>
58  /// </summary>
59  /// <param name="disposal"><para>The <see cref="Disposal"/>
    ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
60  public static implicit operator Disposable(Disposal disposal) => new
    ↳ Disposable(disposal);
61
62  /// <summary>
63  /// <para>Disposes unmanaged resources.</para>
64  /// <para>Высвобождает неуправляемые ресурсы.</para>
65  /// </summary>
66  /// <param name="manual">
67  /// <para>A value that determines whether the disposal was triggered manually (by the
    ↳ developer's code) or was executed automatically without an explicit indication from
    ↳ the developer.</para>
68  /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↳ разработчика.</para>
69  /// </param>
70  /// <param name="wasDisposed">
71  /// <para>A value that determines whether the object was released before calling this
    ↳ method.</para>
72  /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
73  /// </param>
74  protected override void Dispose(bool manual, bool wasDisposed) =>
    ↳ RaiseOnDisposeEvent(manual, wasDisposed);
75
76  /// <summary>
77  /// <para>Raises an unmanaged resource dispose event.</para>
78  /// <para>Генерирует событие высвобождения неуправляемых ресурсов.</para>
79  /// </summary>
80  /// <param name="manual">
81  /// <para>A value that determines whether the disposal was triggered manually (by the
    ↳ developer's code) or was executed automatically without an explicit indication from
    ↳ the developer.</para>
82  /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↳ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↳ разработчика.</para>
83  /// </param>
84  /// <param name="wasDisposed">
85  /// <para>A value that determines whether the object was released before calling this
    ↳ method.</para>
86  /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
87  /// </param>
88  protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
    ↳ wasDisposed);
89
90  /// <summary>
91  /// <para>Attempts to dispose the specified object, as well as set the value of the
    ↳ variable containing this object to the default value.</para>
92  /// <para>Выполняет попытку высвободить указанный объект, а так же установить значение
    ↳ переменной содержащей этот объект в значение по умолчанию.</para>
93  /// </summary>
94  /// <typeparam name="T"><para>Type of the specified object.</para><para>Тип указанного
    ↳ объекта.</para></typeparam>
95  /// <param name="object"><para>The object to dispose.</para><para>Объект, который
    ↳ необходимо высвободить.</para></param>
96  /// <returns><para>A value that determines whether the attempt to release the specified
    ↳ object was successful.</para><para>Значение, определяющие удачно ли была выполнена
    ↳ попытка высвободить указанный объект.</para></returns>
97  public static bool TryDisposeAndResetToDefault<T>(ref T @object)
98  {
99      var result = @object.TryDispose();

```

```

100         if (result)
101         {
102             @object = default;
103         }
104         return result;
105     }
106 }
107 }

```

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

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// <para>Represents disposable container that disposes contained object when the container
7      ///     ↪ itself is disposed.</para>
8      /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
9      ///     ↪ 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 { get; }
18
19         /// <summary>
20         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
21         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
22         /// </summary>
23         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
24         /// <param name="action"><para>The <see cref="Action{T}"> delegate.</para><para>Делегат
25         ///     ↪ <see cref="Action{T}">.</para></param>
26         public Disposable(T @object, Action<T> action)
27         {
28             Object = @object;
29             OnDispose += (manual, wasDisposed) =>
30             {
31                 if (!wasDisposed)
32                 {
33                     action(Object);
34                 }
35             };
36         }
37
38         /// <summary>
39         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
40         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
41         /// </summary>
42         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
43         /// <param name="action"><para>The <see cref="Action{T}"> delegate.</para><para>Делегат
44         ///     ↪ <see cref="Action{T}">.</para></param>
45         public Disposable(T @object, Action action) : base(action) => Object = @object;
46
47         /// <summary>
48         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
49         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
50         /// </summary>
51         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
52         /// <param name="disposal"><para>The <see cref="Disposal{T}">
53         ///     ↪ delegate.</para><para>Делегат <see cref="Disposal{T}">.</para></param>
54         public Disposable(T @object, Disposal disposal) : base(disposal) => Object = @object;
55
56         /// <summary>
57         /// <para>Initializes a new instance of the <see cref="Disposable{T}"> object.</para>
58         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{T}">.</para>
59         /// </summary>
60         /// <param name="object"><para>The object.</para><para>Объект.</para></param>
61         public Disposable(T @object) => Object = @object;
62
63         /// <summary>
64         /// <para>Creates a new <see cref="Disposable{T}"> object initialized with <see
65         ///     ↪ cref="ValueTuple{T, TDisposal}.Item1"/> as <see cref="Disposable{T}.Object"/> and
66         ///     ↪ <see cref="ValueTuple{T, TAction}.Item2"/> as delegate <see
67         ///     ↪ cref="Action{T}">.</para>

```

```

60  /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
    <see cref="ValueTuple{T, TDisposal}.Item1" /> как <see cref="Disposable{T}.Object" />
    &rightarrow; и <see cref="ValueTuple{T, TAction}.Item2" /> как делегат <see
    &rightarrow; cref="Action{T}" />.</para>
61  /// </summary>
62  /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
63  public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
    &rightarrow; Disposable<T>(tuple.Item1, tuple.Item2);
64
65  /// <summary>
66  /// <para>Creates a new <see cref="Disposable{T}" /> object initialized with <see
    &rightarrow; cref="ValueTuple{T, TDisposal}.Item1" /> as <see cref="Disposable{T}.Object" /> and
    &rightarrow; <see cref="ValueTuple{T, TAction}.Item2" /> as delegate <see cref="Action" />.</para>
67  /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
    &rightarrow; <see cref="ValueTuple{T, TDisposal}.Item1" /> как <see cref="Disposable{T}.Object" />
    &rightarrow; и <see cref="ValueTuple{T, TAction}.Item2" /> как делегат <see cref="Action" />.</para>
68  /// </summary>
69  /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
70  public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
    &rightarrow; Disposable<T>(tuple.Item1, tuple.Item2);
71
72  /// <summary>
73  /// <para>Creates a new <see cref="Disposable{T}" /> object initialized with <see
    &rightarrow; cref="ValueTuple{T, TDisposal}.Item1" /> as <see cref="Disposable{T}.Object" /> and
    &rightarrow; <see cref="ValueTuple{T, TDisposal}.Item2" /> as delegate <see
    &rightarrow; cref="Disposal" />.</para>
74  /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
    &rightarrow; <see cref="ValueTuple{T, TDisposal}.Item1" /> как <see cref="Disposable{T}.Object" />
    &rightarrow; и <see cref="ValueTuple{T, TDisposal}.Item2" /> как делегат <see
    &rightarrow; cref="Disposal" />.</para>
75  /// </summary>
76  /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
77  public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
    &rightarrow; Disposable<T>(tuple.Item1, tuple.Item2);
78
79  /// <summary>
80  /// <para>Creates a new <see cref="Disposable{T}" /> object initialized with specified
    &rightarrow; object as <see cref="Disposable{T}.Object" />.</para>
81  /// <para>Создает новый объект <see cref="Disposable{T}" />, инициализированную с помощью
    &rightarrow; указанного объекта как <see cref="Disposable{T}.Object" />.</para>
82  /// </summary>
83  /// <param name="object"><para>The object.</para><para>Объект.</para></param>
84  public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
85
86  /// <summary>
87  /// <para>Creates a new copy of the primary object (<see
    &rightarrow; cref="Disposable{T}.Object" />).</para>
88  /// <para>Создаёт новую копию основного объекта (<see
    &rightarrow; cref="Disposable{T}.Object" />).</para>
89  /// </summary>
90  /// <param name="disposableContainer"><para>The disposable
    &rightarrow; container.</para><para>Высвобождаемый контейнер.</para></param>
91  public static implicit operator T(Disposable<T> disposableContainer) =>
    &rightarrow; disposableContainer.Object;
92
93  /// <summary>
94  /// <para>Disposes unmanaged resources.</para>
95  /// <para>Высвобождает неуправляемые ресурсы.</para>
96  /// </summary>
97  /// <param name="manual">
98  /// <para>A value that determines whether the disposal was triggered manually (by the
    &rightarrow; developer's code) or was executed automatically without an explicit indication from
    &rightarrow; the developer.</para>
99  /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    &rightarrow; разработчика) или же выполнилось автоматически без явного указания со стороны
    &rightarrow; разработчика.</para>
100  /// </param>
101  /// <param name="wasDisposed">
102  /// <para>A value that determines whether the object was released before calling this
    &rightarrow; method.</para>
103  /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
104  /// </param>
105  protected override void Dispose(bool manual, bool wasDisposed)
106  {
107      base.Dispose(manual, wasDisposed);
108      Object.TryDispose();

```

```

109     }
110 }
111 }

```

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

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// <para>Represents disposable container that disposes two contained objects when the
7      ///     ↳ container itself is disposed.</para>
8      /// <para>Представляет высвобождаемый контейнер, который высвобождает два содержащихся в
9      ///     ↳ нём объектов при высвобождении самого контейнера.</para>
10     /// </summary>
11     /// <typeparam name="TPrimary"><para>The primary object type.</para><para>Тип основного
12     ///     ↳ объекта.</para></typeparam>
13     /// <typeparam name="TAuxiliary"><para>The auxiliary object type.</para><para>Тип
14     ///     ↳ вспомогательного объекта.</para></typeparam>
15     public class Disposable<TPrimary, TAuxiliary> : Disposable<TPrimary>
16     {
17         /// <summary>
18         /// <para>Gets the auxiliary object.</para>
19         /// <para>Возвращает вспомогательный объект.</para>
20         /// </summary>
21         public TAuxiliary AuxiliaryObject { get; }
22
23         /// <summary>
24         /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
25         ///     ↳ TAuxiliary}"> object.</para>
26         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
27         ///     ↳ TAuxiliary}">.</para>
28         /// </summary>
29         /// <param name="object"><para>The primary object.</para><para>Основной
30         ///     ↳ объект.</para></param>
31         /// <param name="auxiliaryObject"><para>The auxiliary
32         ///     ↳ object.</para><para>Вспомогательный объект.</para></param>
33         /// <param name="action"><para>The <see cref="Action{TPrimary, TAuxiliary}">
34         ///     ↳ delegate.</para><para>Делегат <see cref="Action{TPrimary,
35         ///     ↳ TAuxiliary}">.</para></param>
36         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPrimary,
37         ///     ↳ TAuxiliary> action)
38         : base(@object)
39         {
40             AuxiliaryObject = auxiliaryObject;
41             OnDispose += (manual, wasDisposed) =>
42             {
43                 if (!wasDisposed)
44                 {
45                     action(Object, AuxiliaryObject);
46                 }
47             };
48         }
49
50         /// <summary>
51         /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
52         ///     ↳ TAuxiliary}"> object.</para>
53         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
54         ///     ↳ TAuxiliary}">.</para>
55         /// </summary>
56         /// <param name="object"><para>The primary object.</para><para>Основной
57         ///     ↳ объект.</para></param>
58         /// <param name="auxiliaryObject"><para>The auxiliary
59         ///     ↳ object.</para><para>Вспомогательный объект.</para></param>
60         /// <param name="action"><para>The <see cref="Action"> delegate.</para><para>Делегат
61         ///     ↳ <see cref="Action">.</para></param>
62         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
63         ///     ↳ base(@object, action) => AuxiliaryObject = auxiliaryObject;
64
65         /// <summary>
66         /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
67         ///     ↳ TAuxiliary}"> object.</para>
68         /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
69         ///     ↳ TAuxiliary}">.</para>
70         /// </summary>
71         /// <param name="object"><para>The primary object.</para><para>Основной
72         ///     ↳ объект.</para></param>

```

```

53 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
54 /// <param name="disposal"><para>The <see cref="Disposal"/>
    ↳ delegate.</para><para>Делегат <see cref="Disposal"/>.</para></param>
55 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
    ↳ base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
56
57 /// <summary>
58 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/> object.</para>
59 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/>.</para>
60 /// </summary>
61 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
62 /// <param name="auxiliaryObject"><para>The auxiliary
    ↳ object.</para><para>Вспомогательный объект.</para></param>
63 public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
    ↳ AuxiliaryObject = auxiliaryObject;
64
65 /// <summary>
66 /// <para>Initializes a new instance of the <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/> object.</para>
67 /// <para>Инициализирует новый экземпляр объекта <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}"/>.</para>
68 /// </summary>
69 /// <param name="object"><para>The primary object.</para><para>Основной
    ↳ объект.</para></param>
70 public Disposable(TPrimary @object) : base(@object) { }
71
72 /// <summary>
73 /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
    ↳ initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1"/> as
    ↳ <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
    ↳ TAuxiliary, TAction}.Item2"/> as <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item3"/> as delegate <see cref="Action{TPrimary, TAuxiliary}"/>.</para>
74 /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
    ↳ инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
    ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2"/> как <see
    ↳ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
    ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3"/> как делегат <see
    ↳ cref="Action{TPrimary, TAuxiliary}"/>.</para>
75 /// </summary>
76 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
77 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
    ↳ TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
78
79 /// <summary>
80 /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"/> object
    ↳ initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item1"/> as
    ↳ <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
    ↳ TAuxiliary, TAction}.Item2"/> as <see cref="Disposable{TPrimary,
    ↳ TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item3"/> as delegate <see cref="Action"/>.</para>
81 /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}"/>,
    ↳ инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↳ TAction}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
    ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item2"/> как <see
    ↳ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
    ↳ cref="ValueTuple{TPrimary, TAuxiliary, TAction}.Item3"/> как делегат <see
    ↳ cref="Action"/>.</para>
82 /// </summary>
83 /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
84 public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↳ TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
    ↳ tuple.Item2, tuple.Item3);
85
86 /// <summary>

```



```

87  /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"> object
    initialized with <see cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item1"/> as
    ↪ <see cref="Disposable{TPrimary}.Object"/>, <see cref="ValueTuple{TPrimary,
    ↪ TAuxiliary, TDisposal}.Item2"/> as <see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/> and <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↪ TDisposal}.Item3"/> as delegate <see cref="Disposal"/>.</para>
88  /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}">,
    инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary,
    ↪ TDisposal}.Item1"/> как <see cref="Disposable{TPrimary}.Object"/>, <see
    ↪ cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item2"/> как <see
    ↪ cref="Disposable{TPrimary, TAuxiliary}.AuxiliaryObject"/> и <see
    ↪ cref="ValueTuple{TPrimary, TAuxiliary, TDisposal}.Item3"/> как делегат <see
    ↪ cref="Disposal"/>.</para>
89  /// </summary>
90  /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
91  public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↪ TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
    ↪ tuple.Item2, tuple.Item3);
92
93  /// <summary>
94  /// <para>Creates a new <see cref="Disposable{TPrimary, TAuxiliary}"> object
    initialized with <see cref="ValueTuple{TPrimary, TAuxiliary}.Item1"/> as <see
    ↪ cref="Disposable{TPrimary}.Object"/> and <see cref="ValueTuple{TPrimary,
    ↪ TAuxiliary}.Item2"/> as <see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/>.</para>
95  /// <para>Создает новый объект <see cref="Disposable{TPrimary, TAuxiliary}">,
    инициализированную с помощью <see cref="ValueTuple{TPrimary, TAuxiliary}.Item1"/>
    ↪ как <see cref="Disposable{TPrimary}.Object"/> и <see cref="ValueTuple{TPrimary,
    ↪ TAuxiliary}.Item2"/> как <see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/>.</para>
96  /// </summary>
97  /// <param name="tuple"><para>The tuple.</para><para>Копреж.</para></param>
98  public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
    ↪ TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
99
100  /// <summary>
101  /// <para>Creates a new copy of the primary object (<see
    ↪ cref="Disposable{TPrimary}.Object"/>).</para>
102  /// <para>Создаёт новую копию основного объекта (<see
    ↪ cref="Disposable{TPrimary}.Object"/>).</para>
103  /// </summary>
104  /// <param name="disposableContainer"><para>The disposable
    ↪ container.</para><para>Высвобождаемый контейнер.</para></param>
105  public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
    ↪ disposableContainer) => disposableContainer.Object;
106
107  /// <summary>
108  /// <para>Creates a new copy of the auxiliary object (<see
    ↪ cref="Disposable{TPrimary}.Object"/>).</para>
109  /// <para>Создаёт новую копию вспомогательного объекта (<see cref="Disposable{TPrimary,
    ↪ TAuxiliary}.AuxiliaryObject"/>).</para>
110  /// </summary>
111  /// <param name="disposableContainer"><para>The disposable
    ↪ container.</para><para>Высвобождаемый контейнер.</para></param>
112  public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
    ↪ disposableContainer) => disposableContainer.AuxiliaryObject;
113
114  /// <summary>
115  /// <para>Disposes unmanaged resources.</para>
116  /// <para>Высвобождает неуправляемые ресурсы.</para>
117  /// </summary>
118  /// <param name="manual">
119  /// <para>A value that determines whether the disposal was triggered manually (by the
    ↪ developer's code) or was executed automatically without an explicit indication from
    ↪ the developer.</para>
120  /// <para>Значение определяющие было ли высвобождение вызвано вручную (кодом
    ↪ разработчика) или же выполнилось автоматически без явного указания со стороны
    ↪ разработчика.</para>
121  /// </param>
122  /// <param name="wasDisposed">
123  /// <para>A value that determines whether the object was released before calling this
    ↪ method.</para>
124  /// <para>Значение определяющие был ли высвобожден объект до вызова этого метода.</para>
125  /// </param>
126  protected override void Dispose(bool manual, bool wasDisposed)

```

```

127     {
128         RaiseOnDisposeEvent(manual, wasDisposed);
129         AuxiliaryObject.TryDispose();
130         Object.TryDispose();
131     }
132 }
133 }

```

#### ./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 }

```

#### ./Platform.Disposables/EnsureExtensions.cs

```

1 using System;
2 using System.Diagnostics;
3 using System.Runtime.CompilerServices;
4 using Platform.Exceptions;
5 using Platform.Exceptions.ExtensionRoots;
6
7 #pragma warning disable IDE0060 // Remove unused parameter
8
9 namespace Platform.Disposables
10 {
11     /// <summary>
12     /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/>
13     /// ↪ and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
14     /// <para>Предоставляет набор методов расширения для объектов <see
15     /// ↪ cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
16     /// </summary>
17     public static class EnsureExtensions
18     {
19         #region Always
20
21         /// <summary>
22         /// <para>Ensures that an object that implements the <see cref="IDisposable"/> interface
23         /// ↪ has not been released. This check is performed regardless of the build
24         /// ↪ configuration.</para>
25         /// <para>Гарантирует, что объект, реализующий интерфейс <see cref="IDisposable"/> не
26         /// ↪ был высвобожден. Эта проверка выполняется независимо от конфигурации
27         /// ↪ сборки.</para>
28         /// </summary>
29         /// <param name="root"><para>The extension root to which this method is
30         /// ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
31         /// <param name="disposable"><para>The object implementing the <see cref="IDisposable"/>
32         /// ↪ interface.</para><para>Объект, реализующий интерфейс <see
33         /// ↪ cref="IDisposable"/></para></param>
34         /// <param name="objectName"><para>The name of object.</para><para>Имя
35         /// ↪ объекта.</para></param>
36         /// <param name="message"><para>The message of the thrown
37         /// ↪ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
40         ↪ disposable, string objectName, string message)
41         {
42             if (disposable.IsDisposed)
43             {
44                 throw new ObjectDisposedException(objectName, message);
45             }
46         }
47     }
48 }

```

```

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 }

```

#### ./Platform.Disposables/GenericObjectExtensions.cs

```

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

```

```

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

```

#### ./Platform.Disposables/IDisposable.cs

```

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

```

#### ./Platform.Disposables/IDisposableExtensions.cs

```

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

```

#### ./Platform.Disposables.Tests/DisposableTests.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Diagnostics;
4 using System.IO;
5 using System.Threading;
6 using Xunit;
7

```

```

8 namespace Platform.Disposables.Tests
9 {
10     public static class DisposableTests
11     {
12         [Fact]
13         public static void DisposalOrderTest()
14         {
15             var path = GetDisposalObjectTestProjectFilePath();
16             var logPath = Path.GetTempFileName();
17             using (var process = Process.Start("dotnet", $"run -p \"{path}\" \"{logPath}\"
18                 ↪ 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 path = GetDisposalObjectTestProjectFilePath();
31             var logPath = Path.GetTempFileName();
32             using (var process = Process.Start("dotnet", $"run -p \"{path}\" \"{logPath}\"
33                 ↪ true"))
34             {
35                 Thread.Sleep(1000);
36                 process.Kill();
37             }
38             var result = File.ReadAllText(logPath);
39             Assert.Equal("", result); // Currently, process termination will not release
40                                     ↪ resources
41             File.Delete(logPath);
42         }
43
44         private static string GetDisposalObjectTestProjectFilePath()
45         {
46             const string currentProjectName = nameof(Platform) + "." + nameof(Disposables) + "."
47                 ↪ + nameof(Tests);
48             const string disposalOrderTestProjectName = currentProjectName + "." +
49                 ↪ nameof(DisposalOrderTest);
50             var currentDirectory = Environment.CurrentDirectory;
51             var pathParts = currentDirectory.Split(Path.DirectorySeparatorChar);
52             var newPathParts = new List<string>();
53             for (var i = 0; i < pathParts.Length; i++)
54             {
55                 if (string.Equals(pathParts[i], currentProjectName))
56                 {
57                     newPathParts.Add(disposalOrderTestProjectName);
58                     break;
59                 }
60                 else
61                 {
62                     newPathParts.Add(pathParts[i]);
63                 }
64             }
65             pathParts = newPathParts.ToArray();
66             var path = Path.Combine(Path.Combine(pathParts),
67                 ↪ $"{disposalOrderTestProjectName}.csproj");
68             if (!Path.IsPathRooted(path))
69             {
70                 path = $"{Path.DirectorySeparatorChar}{path}";
71             }
72             return path;
73         }
74     }
75 }

```

./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>

```

```
7      /// <para>Содержит тесты для функций самого .NET Framework, которые требуются для текущих
8      ↪ реализаций.</para>
9      /// </summary>
10     public static class SystemTests
11     {
12         [Fact]
13         public static void UsingSupportsNullTest()
14         {
15             using (var disposable = null as IDisposable)
16             {
17                 Assert.True(disposable == null);
18             }
19         }
20     }
```

## Index

- ./Platform.Disposables.Tests/DisposableTests.cs, 13
- ./Platform.Disposables.Tests/SystemTests.cs, 14
- ./Platform.Disposables/Disposable.cs, 3
- ./Platform.Disposables/DisposableBase.cs, 1
- ./Platform.Disposables/Disposable[TPrimary, TAuxiliary].cs, 7
- ./Platform.Disposables/Disposable[T].cs, 5
- ./Platform.Disposables/Disposal.cs, 10
- ./Platform.Disposables/EnsureExtensions.cs, 10
- ./Platform.Disposables/GenericObjectExtensions.cs, 12
- ./Platform.Disposables/IDisposable.cs, 13
- ./Platform.Disposables/IDisposableExtensions.cs, 13