

LinksPlatform's Platform.Disposables Class Library

./DisposableBase.cs

```
1  using System;
2  using System.Diagnostics;
3  using System.Threading;
4  using Platform.Exceptions;
5
6  namespace Platform.Disposables
7  {
8      /// <summary>
9      /// Provides a base implementation for IDisposable interface with the basic logic necessary
10     ↪ to increase the likelihood of correct unmanaged resources release.
11     /// Предоставляет базовую реализацию для интерфейса IDisposable с основной логикой
12     ↪ необходимой для повышения вероятности корректного высвобождения неуправляемых ресурсов.
13     /// </summary>
14     public abstract class DisposableBase : IDisposable
15     {
16         private static readonly Process _currentProcess = Process.GetCurrentProcess();
17
18         private volatile int _disposed;
19
20         public bool IsDisposed => _disposed > 0;
21
22         protected virtual string ObjectName => GetType().Name;
23
24         protected virtual bool AllowMultipleDisposeAttempts => false;
25
26         protected virtual bool AllowMultipleDisposeCalls => false;
27
28         protected DisposableBase()
29         {
30             _disposed = 0;
31             _currentProcess.Exited += OnProcessExit;
32         }
33
34         ~DisposableBase() => Destruct();
35
36         protected abstract void Dispose(bool manual, bool wasDisposed);
37
38         public void Dispose()
39         {
40             GC.SuppressFinalize(this);
41             Dispose(true);
42         }
43
44         public void Destruct()
45         {
46             try
47             {
48                 if (!IsDisposed)
49                 {
50                     Dispose(false);
51                 }
52             }
53             catch (Exception exception)
54             {
55                 exception.Ignore();
56             }
57         }
58
59         private void OnProcessExit(object sender, EventArgs e)
60         {
61             GC.SuppressFinalize(this);
62             Destruct();
63         }
64
65         private void Dispose(bool manual)
66         {
67             var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
68             var wasDisposed = originalDisposedValue > 0;
69             if (!wasDisposed)
70             {
71                 UnsubscribeFromProcessExitedEventIfPossible();
72             }
73             if (wasDisposed && !AllowMultipleDisposeCalls && manual)
74             {
75                 Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are now
76                 ↪ allowed. Override AllowMultipleDisposeCalls property to modify behaviour.");
77             }
78             if (AllowMultipleDisposeAttempts || !wasDisposed)
```

```

76         {
77             Dispose(manual, wasDisposed);
78         }
79     }
80
81     private void UnsubscribeFromProcessExitedEventIfPossible()
82     {
83         try
84         {
85             if (_currentProcess != null)
86             {
87                 _currentProcess.Exited -= OnProcessExit;
88             }
89             else
90             {
91                 Process.GetCurrentProcess().Exited -= OnProcessExit;
92             }
93         }
94         catch (Exception exception)
95         {
96             exception.Ignore();
97         }
98     }
99 }
100 }

```

./DisposableBaseExtensions.cs

```

1  namespace Platform.Disposables
2  {
3      public static class DisposableBaseExtensions
4      {
5          public static void DisposeIfNotDisposed(this DisposableBase disposable)
6          {
7              if (!disposable.IsDisposed)
8              {
9                  disposable.Dispose();
10             }
11         }
12     }
13 }

```

./Disposable.cs

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// Represents disposable object that contains OnDispose event which is raised when the
7      /// ↳ object itself is disposed.
8      /// Представляет высвобождаемый объект, который содержит событие OnDispose, которое
9      /// ↳ возникает при высвобождении самого объекта.
10     /// </summary>
11     public class Disposable : DisposableBase
12     {
13         private static readonly Disposal _emptyDelegate = (manual, wasDisposed) => { };
14         public event Disposal OnDispose;
15         public Disposable(Action disposed)
16         {
17             OnDispose = (manual, wasDisposed) =>
18             {
19                 if (!wasDisposed)
20                 {
21                     disposed();
22                 }
23             };
24         }
25
26         public Disposable(Disposal disposed) => OnDispose = disposed;
27         public Disposable() => OnDispose = _emptyDelegate;
28         public static implicit operator Disposable(Action action) => new Disposable(action);
29         public static implicit operator Disposable(Disposal disposal) => new
30         ↳ Disposable(disposal);
31     }
32 }
33

```

```

34     protected override void Dispose(bool manual, bool wasDisposed) => OnDispose(manual,
    ↪     wasDisposed);
35
36     protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
    ↪     wasDisposed);
37
38     public static bool TryDisposeAndResetToDefault<T>(ref T @object)
39     {
40         var result = @object.TryDispose();
41         if (result)
42         {
43             @object = default;
44         }
45         return result;
46     }
47 }
48 }

```

./Disposable[T].cs

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>
6      /// Represents disposable container that disposes contained object when the container itself
    ↪     is disposed.
7      /// Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём объект
    ↪     при высвобождении самого контейнера.
8      /// </summary>
9      public class Disposable<T> : Disposable
10     {
11         public T Object { get; }
12
13         public Disposable(T @object, Action<T> disposed)
14         {
15             Object = @object;
16             OnDispose += (manual, wasDisposed) =>
17             {
18                 if (!wasDisposed)
19                 {
20                     disposed(Object);
21                 }
22             };
23         }
24
25         public Disposable(T @object, Action disposed) : base(disposed) => Object = @object;
26         public Disposable(T @object, Disposal disposed) : base(disposed) => Object = @object;
27         public Disposable(T @object) => Object = @object;
28
29         public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
    ↪     Disposable<T>(tuple.Item1, tuple.Item2);
30
31         public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
    ↪     Disposable<T>(tuple.Item1, tuple.Item2);
32
33         public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
    ↪     Disposable<T>(tuple.Item1, tuple.Item2);
34
35         public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
36
37         public static implicit operator T(Disposable<T> disposable) => disposable.Object;
38
39         protected override void Dispose(bool manual, bool wasDisposed)
40         {
41             base.Dispose(manual, wasDisposed);
42             Object.TryDispose();
43         }
44     }
45 }
46
47 }

```

./Disposable[TPrimary, TAuxiliary].cs

```

1  using System;
2
3  namespace Platform.Disposables
4  {
5      /// <summary>

```

```

6  /// Represents disposable container that disposes two contained objects when the container
   ↳ itself is disposed.
7  /// Представляет высвобождаемый контейнер, который высвобождает два содержащихся в нём
   ↳ объектов при высвобождении самого контейнера.
8  /// </summary>
9  public class Disposable<TPPrimary, TAuxiliary> : Disposable<TPPrimary>
10 {
11     public TAuxiliary AuxiliaryObject { get; }
12
13     public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPPrimary,
   ↳ TAuxiliary> action)
14         : base(@object)
15     {
16         AuxiliaryObject = auxiliaryObject;
17         OnDispose += (manual, wasDisposed) =>
18         {
19             if (!wasDisposed)
20             {
21                 action(Object, AuxiliaryObject);
22             }
23         };
24     }
25
26     public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
   ↳ base(@object, action) => AuxiliaryObject = auxiliaryObject;
27
28     public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
   ↳ base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
29
30     public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
   ↳ AuxiliaryObject = auxiliaryObject;
31
32     public Disposable(TPrimary @object) : base(@object) { }
33
34     public static implicit operator Disposable<TPPrimary, TAuxiliary>(ValueTuple<TPPrimary,
   ↳ TAuxiliary, Action<TPPrimary, TAuxiliary>> tuple) => new Disposable<TPPrimary,
   ↳ TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
35
36     public static implicit operator Disposable<TPPrimary, TAuxiliary>(ValueTuple<TPPrimary,
   ↳ TAuxiliary, Action> tuple) => new Disposable<TPPrimary, TAuxiliary>(tuple.Item1,
   ↳ tuple.Item2, tuple.Item3);
37
38     public static implicit operator Disposable<TPPrimary, TAuxiliary>(ValueTuple<TPPrimary,
   ↳ TAuxiliary, Disposal> tuple) => new Disposable<TPPrimary, TAuxiliary>(tuple.Item1,
   ↳ tuple.Item2, tuple.Item3);
39
40     public static implicit operator Disposable<TPPrimary, TAuxiliary>(ValueTuple<TPPrimary,
   ↳ TAuxiliary> tuple) => new Disposable<TPPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
41
42     public static implicit operator TPrimary(Disposable<TPPrimary, TAuxiliary>
   ↳ disposableContainer) => disposableContainer.Object;
43
44     public static implicit operator TAuxiliary(Disposable<TPPrimary, TAuxiliary>
   ↳ disposableContainer) => disposableContainer.AuxiliaryObject;
45
46     protected override void Dispose(bool manual, bool wasDisposed)
47     {
48         RaiseOnDisposeEvent(manual, wasDisposed);
49         AuxiliaryObject.TryDispose();
50         Object.TryDispose();
51     }
52 }
53 }

```

./Disposal.cs

```

1  namespace Platform.Disposables
2  {
3      public delegate void Disposal(bool manual, bool wasDisposed);
4  }

```

./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     public static class EnsureExtensions
12     {
13         #region Always
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↳ disposable) => NotDisposed(root, disposable, null, null);
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↳ disposable, string objectName) => NotDisposed(root, disposable, objectName, null);
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↳ disposable, string objectName, string message)
23         {
24             if (disposable.IsDisposed)
25             {
26                 throw new ObjectDisposedException(objectName, message);
27             }
28         }
29
30         #endregion
31
32         #region OnDebug
33
34         [Conditional("DEBUG")]
35         public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↳ disposable) => Ensure.Always.NotDisposed(disposable, null, null);
36
37         [Conditional("DEBUG")]
38         public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↳ disposable, string objectName) => Ensure.Always.NotDisposed(disposable, objectName,
            ↳ null);
39
40         [Conditional("DEBUG")]
41         public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↳ disposable, string objectName, string message) =>
            ↳ Ensure.Always.NotDisposed(disposable, objectName, message);
42
43         #endregion
44     }
45 }

```

./GenericObjectExtensions.cs

```

1 using System;
2 using Platform.Exceptions;
3
4 namespace Platform.Disposables
5 {
6     static public class GenericObjectExtensions
7     {
8         public static bool TryDispose<T>(this T @object)
9         {
10             try
11             {
12                 if (@object is DisposableBase disposableBase)
13                 {
14                     disposableBase.DisposeIfNotDisposed();
15                 }
16                 else if (@object is System.IDisposable disposable)
17                 {
18                     disposable.Dispose();
19                 }
20                 return true;
21             }
22             catch (Exception exception)
23             {
24                 exception.Ignore();
25             }
26             return false;
27         }
28
29         public static void DisposeIfPossible<T>(this T @object) => TryDispose(@object);
30     }
31 }

```

./IDisposable.cs

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

Index

- ./Disposable.cs, 2
- ./DisposableBase.cs, 1
- ./DisposableBaseExtensions.cs, 2
- ./Disposable[TPrimary, TAuxiliary].cs, 3
- ./Disposable[T].cs, 3
- ./Disposal.cs, 4
- ./EnsureExtensions.cs, 4
- ./GenericObjectExtensions.cs, 5
- ./IDisposable.cs, 6