

LinksPlatform's Platform.Disposables Class Library

./Platform.Disposables/DisposableBase.cs

```
1  using System;
2  using System.Diagnostics;
3  using System.Threading;
4  using Platform.Exceptions;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Disposables
9  {
10     /// <summary>
11     /// <para>Provides a base implementation for IDisposable interface with the basic logic
12     ///     necessary to increase the likelihood of correct unmanaged resources release.</para>
13     /// <para>Предоставляет базовую реализацию для интерфейса IDisposable с основной логикой
14     ///     необходимой для повышения вероятности корректного высвобождения неуправляемых
15     ///     ресурсов.</para>
16     /// </summary>
17     public abstract class DisposableBase : IDisposable
18     {
19         private static readonly Process _currentProcess = Process.GetCurrentProcess();
20
21         private volatile int _disposed;
22
23         public bool IsDisposed => _disposed > 0;
24
25         protected virtual string ObjectName => GetType().Name;
26
27         protected virtual bool AllowMultipleDisposeAttempts => false;
28
29         protected virtual bool AllowMultipleDisposeCalls => false;
30
31         protected DisposableBase()
32         {
33             _disposed = 0;
34             _currentProcess.Exited += OnProcessExit;
35         }
36
37         ~DisposableBase() => Destruct();
38
39         protected abstract void Dispose(bool manual, bool wasDisposed);
40
41         public void Dispose()
42         {
43             GC.SuppressFinalize(this);
44             Dispose(true);
45         }
46
47         public void Destruct()
48         {
49             try
50             {
51                 if (!IsDisposed)
52                 {
53                     Dispose(false);
54                 }
55             }
56             catch (Exception exception)
57             {
58                 exception.Ignore();
59             }
60         }
61
62         private void OnProcessExit(object sender, EventArgs e)
63         {
64             GC.SuppressFinalize(this);
65             Destruct();
66         }
67
68         private void Dispose(bool manual)
69         {
70             var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
71             var wasDisposed = originalDisposedValue > 0;
72             if (!wasDisposed)
73             {
74                 UnsubscribeFromProcessExitedEventIfPossible();
75             }
76             if (wasDisposed && !AllowMultipleDisposeCalls && manual)
77             {
78             
```

```

75         Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are now
76         ↪ allowed. Override AllowMultipleDisposeCalls property to modify behaviour.");
77     }
78     if (AllowMultipleDisposeAttempts || !wasDisposed)
79     {
80         Dispose(manual, wasDisposed);
81     }
82 }
83 private void UnsubscribeFromProcessExitedEventIfPossible()
84 {
85     try
86     {
87         if (_currentProcess != null)
88         {
89             _currentProcess.Exited -= OnProcessExit;
90         }
91         else
92         {
93             Process.GetCurrentProcess().Exited -= OnProcessExit;
94         }
95     }
96     catch (Exception exception)
97     {
98         exception.Ignore();
99     }
100 }
101 }
102 }

```

./Platform.Disposables/DisposableBaseExtensions.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Disposables
4  {
5      public static class DisposableBaseExtensions
6      {
7          public static void DisposeIfNotDisposed(this DisposableBase disposable)
8          {
9              if (!disposable.IsDisposed)
10             {
11                 disposable.Dispose();
12             }
13         }
14     }
15 }

```

./Platform.Disposables/Disposable.cs

```

1  using System;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Disposables
6  {
7      /// <summary>
8      /// <para>Represents disposable object that contains OnDispose event which is raised when
9      ↪ the object itself is disposed.</para>
10     /// <para>Представляет высвобождаемый объект, который содержит событие OnDispose, которое
11     ↪ возникает при высвобождении самого объекта.</para>
12     /// </summary>
13     public class Disposable : DisposableBase
14     {
15         private static readonly Disposal _emptyDelegate = (manual, wasDisposed) => { };
16
17         public event Disposal OnDispose;
18
19         public Disposable(Action disposed)
20         {
21             OnDispose = (manual, wasDisposed) =>
22             {
23                 if (!wasDisposed)
24                 {
25                     disposed();
26                 }
27             };
28         }
29
30         public Disposable(Disposal disposed) => OnDispose = disposed;
31     }
32 }

```

```

29
30     public Disposable() => OnDispose = _emptyDelegate;
31
32     public static implicit operator Disposable(Action action) => new Disposable(action);
33
34     public static implicit operator Disposable(Disposal disposal) => new
35         ↳ Disposable(disposal);
36
37     protected override void Dispose(bool manual, bool wasDisposed) => OnDispose(manual,
38         ↳ wasDisposed);
39
40     protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
41         ↳ wasDisposed);
42
43     public static bool TryDisposeAndResetToDefault<T>(ref T @object)
44     {
45         var result = @object.TryDispose();
46         if (result)
47         {
48             @object = default;
49         }
50         return result;
51     }
52 }

```

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

```

1  using System;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Disposables
6  {
7      /// <summary>
8      /// <para>Represents disposable container that disposes contained object when the container
9      /// ↳ itself is disposed.</para>
10     /// <para>Представляет высвобождаемый контейнер, который высвобождает содержащийся в нём
11     /// ↳ объект при высвобождении самого контейнера.</para>
12     /// </summary>
13     public class Disposable<T> : Disposable
14     {
15         public T Object { get; }
16
17         public Disposable(T @object, Action<T> disposed)
18         {
19             Object = @object;
20             OnDispose += (manual, wasDisposed) =>
21             {
22                 if (!wasDisposed)
23                 {
24                     disposed(Object);
25                 }
26             };
27         }
28
29         public Disposable(T @object, Action disposed) : base(disposed) => Object = @object;
30
31         public Disposable(T @object, Disposal disposed) : base(disposed) => Object = @object;
32
33         public Disposable(T @object) => Object = @object;
34
35         public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
36             ↳ Disposable<T>(tuple.Item1, tuple.Item2);
37
38         public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
39             ↳ Disposable<T>(tuple.Item1, tuple.Item2);
40
41         public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
42             ↳ Disposable<T>(tuple.Item1, tuple.Item2);
43
44         public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);
45
46         public static implicit operator T(Disposable<T> disposable) => disposable.Object;
47
48         protected override void Dispose(bool manual, bool wasDisposed)
49         {
50             base.Dispose(manual, wasDisposed);
51             Object.TryDispose();
52         }
53     }
54 }

```

```

48     }
49 }

```

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

```

1  using System;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Disposables
6  {
7      /// <summary>
8      /// <para>Represents disposable container that disposes two contained objects when the
9      /// <para>Представляет высвобождаемый контейнер, который высвобождает два содержащихся в
10     /// <para>нём объектов при высвобождении самого контейнера.</para>
11     /// </summary>
12     public class Disposable<TPrimary, TAuxiliary> : Disposable<TPrimary>
13     {
14         public TAuxiliary AuxiliaryObject { get; }
15
16         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPrimary,
17             TAuxiliary> action)
18             : base(@object)
19         {
20             AuxiliaryObject = auxiliaryObject;
21             OnDispose += (manual, wasDisposed) =>
22             {
23                 if (!wasDisposed)
24                 {
25                     action(Object, AuxiliaryObject);
26                 }
27             };
28
29             public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
30                 base(@object, action) => AuxiliaryObject = auxiliaryObject;
31
32             public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
33                 base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
34
35             public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
36                 AuxiliaryObject = auxiliaryObject;
37
38             public Disposable(TPrimary @object) : base(@object) { }
39
40             public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
41                 TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
42                 TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
43
44             public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
45                 TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
46                 tuple.Item2, tuple.Item3);
47
48             public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
49                 TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
50                 tuple.Item2, tuple.Item3);
51
52             public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
53                 TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
54
55             public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
56                 disposableContainer) => disposableContainer.Object;
57
58             public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
59                 disposableContainer) => disposableContainer.AuxiliaryObject;
60
61             protected override void Dispose(bool manual, bool wasDisposed)
62             {
63                 RaiseOnDisposeEvent(manual, wasDisposed);
64                 AuxiliaryObject.TryDispose();
65                 Object.TryDispose();
66             }
67         }
68     }
69 }

```

./Platform.Disposables/Disposal.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2

```

```

3 namespace Platform.Disposables
4 {
5     public delegate void Disposal(bool manual, bool wasDisposed);
6 }

```

./Platform.Disposables/EnsureExtensions.cs

```

1 using System;
2 using System.Diagnostics;
3 using System.Runtime.CompilerServices;
4 using Platform.Exceptions;
5 using Platform.Exceptions.ExtensionRoots;
6
7 #pragma warning disable IDE0060 // Remove unused parameter
8 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Disposables
11 {
12     public static class EnsureExtensions
13     {
14         #region Always
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↳ disposable) => NotDisposed(root, disposable, null, null);
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↳ disposable, string objectName) => NotDisposed(root, disposable, objectName, null);
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public static void NotDisposed(this EnsureAlwaysExtensionRoot root, IDisposable
            ↳ disposable, string objectName, string message)
24         {
25             if (disposable.IsDisposed)
26             {
27                 throw new ObjectDisposedException(objectName, message);
28             }
29         }
30
31         #endregion
32
33         #region OnDebug
34
35         [Conditional("DEBUG")]
36         public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↳ disposable) => Ensure.Always.NotDisposed(disposable, null, null);
37
38         [Conditional("DEBUG")]
39         public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↳ disposable, string objectName) => Ensure.Always.NotDisposed(disposable, objectName,
            ↳ null);
40
41         [Conditional("DEBUG")]
42         public static void NotDisposed(this EnsureOnDebugExtensionRoot root, IDisposable
            ↳ disposable, string objectName, string message) =>
            ↳ Ensure.Always.NotDisposed(disposable, objectName, message);
43
44         #endregion
45     }
46 }

```

./Platform.Disposables/GenericObjectExtensions.cs

```

1 using System;
2 using Platform.Exceptions;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Disposables
7 {
8     /// <summary>
9     /// <para>Provides a set of static methods that help dispose an object.</para>
10    /// <para>Предоставляет набор статических методов которые помогают высвободить объект.</para>
11    /// </summary>
12    static public class GenericObjectExtensions
13    {
14        public static bool TryDispose<T>(this T @object)
15        {
16            try
17            {

```

```

18         if (@Object is DisposableBase disposableBase)
19         {
20             disposableBase.DisposeIfNotDisposed();
21         }
22         else if (@Object is System.IDisposable disposable)
23         {
24             disposable.Dispose();
25         }
26         return true;
27     }
28     catch (Exception exception)
29     {
30         exception.Ignore();
31     }
32     return false;
33 }
34
35 public static void DisposeIfPossible<T> (this T @object) => TryDispose(@object);
36
37 }

```

./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.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 самой .NET Framework, которые требуются для текущих
9     ///   ↳ реализаций.</para>
10    /// </summary>
11    public class SystemTests
12    {
13        [Fact]
14        public void UsingSupportsNullTest()
15        {
16            using (var disposable = null as IDisposable)
17            {
18                Assert.True(disposable == null);
19            }
20        }
21    }
22 }

```

Index

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