

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  #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 AppDomain _currentDomain = AppDomain.CurrentDomain;
20         private static readonly ConcurrentStack<WeakReference<DisposableBase>>
21             ↪ _disposablesWeekReferencesStack = new
22             ↪ ConcurrentStack<WeakReference<DisposableBase>>();
23
24         private volatile int _disposed;
25
26         public bool IsDisposed => _disposed > 0;
27
28         protected virtual string ObjectName => GetType().Name;
29
30         protected virtual bool AllowMultipleDisposeAttempts => false;
31
32         protected virtual bool AllowMultipleDisposeCalls => false;
33
34         static DisposableBase() => _currentDomain.ProcessExit += OnProcessExit;
35
36         protected DisposableBase()
37         {
38             _disposed = 0;
39             _disposablesWeekReferencesStack.Push(new WeakReference<DisposableBase>(this, false));
40         }
41
42         ~DisposableBase() => Destruct();
43
44         protected abstract void Dispose(bool manual, bool wasDisposed);
45
46         public void Dispose()
47         {
48             GC.SuppressFinalize(this);
49             Dispose(true);
50         }
51
52         public void Destruct()
53         {
54             try
55             {
56                 if (!IsDisposed)
57                 {
58                     Dispose(false);
59                 }
60             }
61             catch (Exception exception)
62             {
63                 exception.Ignore();
64             }
65         }
66
67         private void Dispose(bool manual)
68         {
69             var originalDisposedValue = Interlocked.CompareExchange(ref _disposed, 1, 0);
70             var wasDisposed = originalDisposedValue > 0;
71             if (wasDisposed && !AllowMultipleDisposeCalls && manual)
72             {
73                 Ensure.Always.NotDisposed(this, ObjectName, "Multiple dispose calls are now
74                 ↪ allowed. Override AllowMultipleDisposeCalls property to modify behaviour.");
75             }
76             if (AllowMultipleDisposeAttempts || !wasDisposed)
77             {
78                 Dispose(manual, wasDisposed);
79             }
80         }
81     }
82 }
```

```

73     }
74 }
75
76 private static void OnProcessExit(object sender, EventArgs e)
77 {
78     while (_disposablesWeekReferencesStack.TryPop(out WeakReference<DisposableBase>
79         ↪ weakReference))
80     {
81         if (weakReference.TryGetTarget(out DisposableBase disposable))
82         {
83             GC.SuppressFinalize(disposable);
84             disposable.Destruct();
85         }
86     }
87     UnsubscribeFromProcessExitedEventIfPossible();
88 }
89
90 private static void UnsubscribeFromProcessExitedEventIfPossible()
91 {
92     try
93     {
94         if (_currentDomain != null)
95         {
96             _currentDomain.ProcessExit -= OnProcessExit;
97         }
98         else
99         {
100             AppDomain.CurrentDomain.ProcessExit -= OnProcessExit;
101         }
102     }
103     catch (Exception exception)
104     {
105         exception.Ignore();
106     }
107 }
108 }

```

./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             {

```

```

21         if (!wasDisposed)
22         {
23             disposed();
24         }
25     };
26 }
27
28 public Disposable(Disposal disposed) => OnDispose = disposed;
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
    ↳ Disposable(disposal);
35
36 protected override void Dispose(bool manual, bool wasDisposed) => OnDispose(manual,
    ↳ wasDisposed);
37
38 protected void RaiseOnDisposeEvent(bool manual, bool wasDisposed) => OnDispose(manual,
    ↳ wasDisposed);
39
40 public static bool TryDisposeAndResetToDefault<T>(ref T @object)
41 {
42     var result = @object.TryDispose();
43     if (result)
44     {
45         @object = default;
46     }
47     return result;
48 }
49 }
50 }

```

./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            public Disposable(T @object, Action disposed) : base(disposed) => Object = @object;
29            public Disposable(T @object, Disposal disposed) : base(disposed) => Object = @object;
30            public Disposable(T @object) => Object = @object;
31
32            public static implicit operator Disposable<T>(ValueTuple<T, Action<T>> tuple) => new
33                ↳ Disposable<T>(tuple.Item1, tuple.Item2);
34
35            public static implicit operator Disposable<T>(ValueTuple<T, Action> tuple) => new
36                ↳ Disposable<T>(tuple.Item1, tuple.Item2);
37
38            public static implicit operator Disposable<T>(ValueTuple<T, Disposal> tuple) => new
39                ↳ Disposable<T>(tuple.Item1, tuple.Item2);
40
41            public static implicit operator Disposable<T>(T @object) => new Disposable<T>(@object);

```

```

40
41     public static implicit operator T(Disposable<T> disposable) => disposable.Object;
42
43     protected override void Dispose(bool manual, bool wasDisposed)
44     {
45         base.Dispose(manual, wasDisposed);
46         Object.TryDispose();
47     }
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      ///     ↪ container itself is disposed.</para>
10     /// <para>Представляет высвобождаемый контейнер, который высвобождает два содержащихся в
11     ///     ↪ нём объектов при высвобождении самого контейнера.</para>
12     /// </summary>
13     public class Disposable<TPrimary, TAuxiliary> : Disposable<TPrimary>
14     {
15         public TAuxiliary AuxiliaryObject { get; }
16
17         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action<TPrimary,
18         ↪ TAuxiliary> action)
19             : base(@object)
20         {
21             AuxiliaryObject = auxiliaryObject;
22             OnDispose += (manual, wasDisposed) =>
23             {
24                 if (!wasDisposed)
25                 {
26                     action(Object, AuxiliaryObject);
27                 }
28             };
29         }
30
31         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Action action) :
32         ↪ base(@object, action) => AuxiliaryObject = auxiliaryObject;
33
34         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject, Disposal disposal) :
35         ↪ base(@object, disposal) => AuxiliaryObject = auxiliaryObject;
36
37         public Disposable(TPrimary @object, TAuxiliary auxiliaryObject) : base(@object) =>
38         ↪ AuxiliaryObject = auxiliaryObject;
39
40         public Disposable(TPrimary @object) : base(@object) { }
41
42         public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
43         ↪ TAuxiliary, Action<TPrimary, TAuxiliary>> tuple) => new Disposable<TPrimary,
44         ↪ TAuxiliary>(tuple.Item1, tuple.Item2, tuple.Item3);
45
46         public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
47         ↪ TAuxiliary, Action> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
48         ↪ tuple.Item2, tuple.Item3);
49
50         public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
51         ↪ TAuxiliary, Disposal> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1,
52         ↪ tuple.Item2, tuple.Item3);
53
54         public static implicit operator Disposable<TPrimary, TAuxiliary>(ValueTuple<TPrimary,
55         ↪ TAuxiliary> tuple) => new Disposable<TPrimary, TAuxiliary>(tuple.Item1, tuple.Item2);
56
57         public static implicit operator TPrimary(Disposable<TPrimary, TAuxiliary>
58         ↪ disposableContainer) => disposableContainer.Object;
59
60         public static implicit operator TAuxiliary(Disposable<TPrimary, TAuxiliary>
61         ↪ disposableContainer) => disposableContainer.AuxiliaryObject;
62
63         protected override void Dispose(bool manual, bool wasDisposed)
64         {
65             RaiseOnDisposeEvent(manual, wasDisposed);
66             AuxiliaryObject.TryDispose();
67         }
68     }
69 }

```

```

52         Object.TryDispose();
53     }
54 }
55 }

```

./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/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             string path = GetDisposalObjectTestProjectFilePath();
16             var logPath = Path.GetTempFileName();
17             using (var process = Process.Start("dotnet", $"run -p \"{path}\" \"{logPath}\""
18                 ↪ false))

```

```

18     {
19         process.WaitForExit();
20     }
21     var result = File.ReadAllText(logPath);
22     Assert.Equal("21", result);
23     File.Delete(logPath);
24 }
25
26 [Fact]
27 public static void DisposalAtProcessKillTest()
28 {
29     string path = GetDisposalObjectTestProjectFilePath();
30     var logPath = Path.GetTempFileName();
31     using (var process = Process.Start("dotnet", $"run -p \"{path}\" \"{logPath}\"
    ↪ true"))
32     {
33         Thread.Sleep(1000);
34         process.Kill();
35     }
36     var result = File.ReadAllText(logPath);
37     Assert.Equal("", result); // Currently process termination will not release resources
38     File.Delete(logPath);
39 }
40
41 private static string GetDisposalObjectTestProjectFilePath()
42 {
43     const string currentProjectName = "Platform.Disposables.Tests";
44     const string disposalOrderTestProjectName =
    ↪ "Platform.Disposables.Tests.DisposalOrderTest";
45     var currentDirectory = Environment.CurrentDirectory;
46     var pathParts = currentDirectory.Split(Path.DirectorySeparatorChar);
47     var newPathParts = new List<string>();
48     for (var i = 0; i < pathParts.Length; i++)
49     {
50         if (string.Equals(pathParts[i], currentProjectName))
51         {
52             newPathParts.Add(disposalOrderTestProjectName);
53             break;
54         }
55         else
56         {
57             newPathParts.Add(pathParts[i]);
58         }
59     }
60     pathParts = newPathParts.ToArray();
61     var path = Path.Combine(Path.Combine(pathParts),
    ↪ $"{disposalOrderTestProjectName}.csproj");
62     if (!Path.IsPathRooted(path))
63     {
64         path = $"{Path.DirectorySeparatorChar}{path}";
65     }
66     return path;
67 }
68 }
69 }

```

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

```


Index

- ./Platform.Disposables.Tests/DisposableTests.cs, 6
- ./Platform.Disposables.Tests/SystemTests.cs, 7
- ./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, 5
- ./Platform.Disposables/EnsureExtensions.cs, 5
- ./Platform.Disposables/GenericObjectExtensions.cs, 5
- ./Platform.Disposables/IDisposable.cs, 6