

LinksPlatform's Platform.Exceptions Class Library

./Platform.Exceptions/Ensure.cs

```

1  using Platform.Exceptions.ExtensionRoots;
2
3  namespace Platform.Exceptions
4  {
5      /// <summary>
6      /// <para>Contains two extensible classes instances that can be supplemented with static
7      ///     ↪ helper methods by using the extension mechanism. These methods ensure the contract
8      ///     ↪ compliance.</para>
9      /// <para>Содержит два экземпляра расширяемых класса, которые можно дополнять статическими
10     ↪ вспомогательными методами путём использования механизма расширений. Эти методы
11     ↪ занимаются гарантированием соответствия контракту.</para>
12     /// </summary>
13     public static class Ensure
14     {
15         /// <summary>
16         /// <para>Gets an instance of the extension root class that contains helper methods to
17         ///     ↪ guarantee compliance with the contract.</para>
18         /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
19         ///     ↪ методы для гарантирования соответствия контракту.</para>
20         /// </summary>
21         public static readonly EnsureAlwaysExtensionRoot Always = new
22             ↪ EnsureAlwaysExtensionRoot();
23
24         /// <summary>
25         /// <para>Gets an instance of the extension root class that contains helper methods to
26         ///     ↪ guarantee compliance with the contract, but are executed only during
27         ///     ↪ debugging.</para>
28         /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
29         ///     ↪ методы для гарантирования соответствия контракту, но выполняются только во время
30         ///     ↪ отладки.</para>
31         /// </summary>
32         public static readonly EnsureOnDebugExtensionRoot OnDebug = new
33             ↪ EnsureOnDebugExtensionRoot();
34     }
35 }

```

./Platform.Exceptions/EnsureExtensions.cs

```

1  using System;
2  using System.Diagnostics;
3  using System.Runtime.CompilerServices;
4  using Platform.Exceptions.ExtensionRoots;
5
6  #pragma warning disable IDE0060 // Remove unused parameter
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Exceptions
10 {
11     public static class EnsureAlwaysExtensions
12     {
13         #region Always
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
17             ↪ TArgument argument, string argumentName, string message)
18             where TArgument : class
19         {
20             if (argument == null)
21             {
22                 throw new ArgumentNullException(argumentName, message);
23             }
24         }
25
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
28             ↪ TArgument argument, string argumentName) where TArgument : class =>
29             ↪ ArgumentNotNull(root, argument, argumentName, $"Argument {argumentName} is null.");
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
33             ↪ TArgument argument) where TArgument : class => ArgumentNotNull(root, argument, null);
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
37             ↪ Predicate<TArgument> predicate, TArgument argument, string argumentName, string
38             ↪ message)
39         {
40
41         }
42     }
43 }

```

```

34         if (!predicate(argument))
35         {
36             throw new ArgumentException(argumentName, message);
37         }
38     }
39
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
42         Predicate<TArgument> predicate, TArgument argument, string argumentName) =>
43         ArgumentMeetsCriteria(root, predicate, argument, argumentName, $"Argument
44         {argumentName} is does not meet criteria.");
45
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
48         Predicate<TArgument> predicate, TArgument argument) => ArgumentMeetsCriteria(root,
49         predicate, argument, null);
50
51     #endregion
52
53     #region OnDebug
54
55     [Conditional("DEBUG")]
56     public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
57         TArgument argument, string argumentName, string message) where TArgument : class =>
58         Ensure.Always.ArgumentNotNull(argument, argumentName, message);
59
60     [Conditional("DEBUG")]
61     public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
62         TArgument argument, string argumentName) where TArgument : class =>
63         Ensure.Always.ArgumentNotNull(argument, argumentName);
64
65     [Conditional("DEBUG")]
66     public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
67         TArgument argument) where TArgument : class =>
68         Ensure.Always.ArgumentNotNull(argument);
69
70     [Conditional("DEBUG")]
71     public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
72         root, Predicate<TArgument> predicate, TArgument argument, string argumentName,
73         string message) => Ensure.Always.ArgumentMeetsCriteria(predicate, argument,
74         argumentName, message);
75
76     [Conditional("DEBUG")]
77     public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
78         root, Predicate<TArgument> predicate, TArgument argument, string argumentName) =>
79         Ensure.Always.ArgumentMeetsCriteria(predicate, argument, argumentName);
80
81     [Conditional("DEBUG")]
82     public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
83         root, Predicate<TArgument> predicate, TArgument argument) =>
84         Ensure.Always.ArgumentMeetsCriteria(predicate, argument);
85
86     #endregion
87 }
88 }

```

./Platform.Exceptions/ExceptionExtensions.cs

```

1  using System;
2  using System.Text;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Exceptions
7  {
8      public static class ExceptionExtensions
9      {
10         public static readonly string ExceptionContentsSeparator = "---";
11         public static readonly string ExceptionStringBuildingFailed = "Unable to format
12         exception.";
13
14         public static void Ignore(this Exception exception) =>
15             IgnoredExceptions.RaiseExceptionIgnoredEvent(exception);
16
17         public static string ToStringWithAllInnerExceptions(this Exception exception)
18         {
19             try
20             {
21                 var sb = new StringBuilder();

```

```

20         BuildExceptionString(sb, exception, 0);
21         return sb.ToString();
22     }
23     catch (Exception ex)
24     {
25         ex.Ignore();
26         return ExceptionStringBuilderFailed;
27     }
28 }
29
30 private static void BuildExceptionString(StringBuilder sb, Exception exception, int
↪ level)
31 {
32     sb.Append('\t', level);
33     sb.Append("Exception message: ");
34     sb.AppendLine(exception.Message);
35     sb.Append('\t', level);
36     sb.AppendLine(ExceptionContentsSeparator);
37     if (exception.InnerException != null)
38     {
39         sb.Append('\t', level);
40         sb.AppendLine("Inner Exception: ");
41         BuildExceptionString(sb, exception.InnerException, level + 1);
42     }
43     sb.Append('\t', level);
44     sb.AppendLine(ExceptionContentsSeparator);
45     sb.Append('\t', level);
46     sb.AppendLine(exception.StackTrace);
47 }
48 }
49 }

```

./Platform.Exceptions/ExtensionRoots/EnsureAlwaysExtensionRoot.cs

```

1 namespace Platform.Exceptions.ExtensionRoots
2 {
3     /// <summary>
4     /// <para>Represents the extension root class for Ensure.Always.</para>
5     /// <para>Представляет класс корень-расширения для Ensure.Always.</para>
6     /// </summary>
7     public class EnsureAlwaysExtensionRoot
8     {
9     }
10 }

```

./Platform.Exceptions/ExtensionRoots/EnsureOnDebugExtensionRoot.cs

```

1 namespace Platform.Exceptions.ExtensionRoots
2 {
3     /// <summary>
4     /// <para>Represents the extension root class for Ensure.OnDebug.</para>
5     /// <para>Представляет класс корень-расширения для Ensure.OnDebug.</para>
6     /// </summary>
7     public class EnsureOnDebugExtensionRoot
8     {
9     }
10 }

```

./Platform.Exceptions/ExtensionRoots/ThrowExtensionRoot.cs

```

1 namespace Platform.Exceptions.ExtensionRoots
2 {
3     /// <summary>
4     /// <para>Represents the extension root class for Throw.A.</para>
5     /// <para>Представляет класс корень-расширения для Throw.A.</para>
6     /// </summary>
7     public class ThrowExtensionRoot
8     {
9     }
10 }

```

./Platform.Exceptions/IgnoredExceptions.cs

```

1 using System;
2 using System.Collections.Concurrent;
3 using System.Collections.Generic;
4
5 namespace Platform.Exceptions
6 {
7     /// <summary>
8     /// <para>Contains a mechanism for notifying about the occurrence of ignored exceptions, as
↪ wellas a mechanism for their collection.</para>

```

```

9      /// <para>Содержит механизм уведомления о возникновении игнорируемых исключений, а так же
    ↳ механизм их сбора.</para>
10     /// </summary>
11     public static class IgnoredExceptions
12     {
13         private static readonly ConcurrentBag<Exception> _exceptionsBag = new
            ↳ ConcurrentBag<Exception>();
14
15         /// <summary>
16         /// <para>An event that is raised every time an exception has been ignored.</para>
17         /// <para>Событие, которое генерируется каждый раз, когда исключение было
            ↳ проигнорировано.</para>
18         /// </summary>
19         public static event EventHandler<Exception> ExceptionIgnored = OnExceptionIgnored;
20
21         /// <summary>
22         /// <para>Gets an immutable collection with all collected exceptions that were
            ↳ ignored.</para>
23         /// <para>Возвращает неизменяемую коллекцию со всеми собранными исключениями которые
            ↳ были проигнорированы.</para>
24         /// </summary>
25         public static IReadOnlyCollection<Exception> CollectedExceptions => _exceptionsBag;
26
27         /// <summary>
28         /// <para>Gets or sets a value that determines whether to collect ignored exceptions
            ↳ into CollectedExceptions.</para>
29         /// <para>Возвращает или устанавливает значение, определяющие нужно ли собирать
            ↳ игнорируемые исключения в CollectedExceptions.</para>
30         /// </summary>
31         public static bool CollectExceptions { get; set; }
32
33         /// <summary>
34         /// <para>Raises an exception ignored event.</para>
35         /// <para>Генерирует событие игнорирования исключения.</para>
36         /// </summary>
37         /// <param name="exception"><para>The ignored exception.</para><para>Игнорируемое
            ↳ исключение.</para></param>
38         /// <remarks>
39         /// <para>It is recommended to call this method in cases where you have a catch block,
            ↳ but you do not do anything with exception in it.</para>
40         /// <para>Рекомендуется вызывать этот метод в тех случаях, когда у вас есть catch блок,
            ↳ но вы ничего не делаете в нём с исключением.</para>
41         /// </remarks>
42         public static void RaiseExceptionIgnoredEvent(Exception exception) =>
            ↳ ExceptionIgnored.Invoke(null, exception);
43
44         private static void OnExceptionIgnored(object sender, Exception exception)
45         {
46             if (CollectExceptions)
47             {
48                 _exceptionsBag.Add(exception);
49             }
50         }
51     }
52 }

```

./Platform.Exceptions/Throw.cs

```

1  using Platform.Exceptions.ExtensionRoots;
2
3  namespace Platform.Exceptions
4  {
5      /// <summary>
6      /// <para>Contains an instance of an extensible class that can be supplemented with static
            ↳ helper methods by using the extension mechanism. These methods throw exceptions.</para>
7      /// <para>Содержит экземпляр расширяемого класса, который можно дополнять статическими
            ↳ вспомогательными методами путём использования механизма расширений. Эти методы
            ↳ занимаются выбрасыванием исключений.</para>
8      /// </summary>
9      public static class Throw
10     {
11         /// <summary>
12         /// <para>Gets an instance of the extension root class that contains helper methods for
            ↳ throwing exceptions.</para>
13         /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
            ↳ методы для выбрасывания исключений.</para>
14         /// </summary>
15         public static readonly ThrowExtensionRoot A = new ThrowExtensionRoot();

```

```
16     }
17 }
```

./Platform.Exceptions/ThrowExtensions.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Exceptions.ExtensionRoots;
4
5 #pragma warning disable IDE0060 // Remove unused parameter
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Exceptions
9 {
10     public static class ThrowExtensions
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static void NotSupportedException(this ThrowExtensionRoot root) => throw new
14             ↳ NotSupportedException();
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static T NotSupportedExceptionAndReturn<T>(this ThrowExtensionRoot root) => throw
18             ↳ new NotSupportedException();
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public static void NotImplementedException(this ThrowExtensionRoot root) => throw new
22             ↳ NotImplementedException();
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public static T NotImplementedExceptionAndReturn<T>(this ThrowExtensionRoot root) =>
26             ↳ throw new NotImplementedException();
27     }
28 }
```

./Platform.Exceptions.Tests/EnsuranceTests.cs

```
1 using System;
2 using Xunit;
3
4 namespace Platform.Exceptions.Tests
5 {
6     public static class EnsuranceTests
7     {
8         [Fact]
9         public static void ArgumentNotNullEnsuranceTest()
10         {
11             // Should throw an exception (even if in neighbour "Ignore" namespace it was
12             ↳ overridden, but here this namespace is not used)
13             Assert.Throws<ArgumentNullException>(() =>
14                 ↳ Ensure.Always.ArgumentNotNull<object>(null, "object"));
15         }
16     }
17 }
```

./Platform.Exceptions.Tests/Ignore/EnsureAlwaysExtensions.cs

```
1 using System.Diagnostics;
2 using Platform.Exceptions.ExtensionRoots;
3
4 namespace Platform.Exceptions.Tests.Ignore
5 {
6     public static class EnsureAlwaysExtensions
7     {
8         [Conditional("DEBUG")]
9         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
10             ↳ TArgument argument, string argumentName)
11             where TArgument : class
12         {
13         }
14     }
15 }
```

./Platform.Exceptions.Tests/Ignore/IgnoredEnsuranceTests.cs

```
1 using Xunit;
2
3 namespace Platform.Exceptions.Tests.Ignore
4 {
5     public static class IgnoredEnsuranceTests
6     {
7         [Fact]
8         public static void EnsuranceIgnoredTest()
```

```
9      {
10          // Should not throw an exception (because logic is overridden in
11          ↪ EnsureAlwaysExtensions that is located within the same namespace)
12          // And even should be optimized out at RELEASE (because method is now marked
13          ↪ conditional DEBUG)
14          // This can be useful in performance critical situations there even an check for
15          ↪ exception is hurting performance enough
16          Ensure.Always.ArgumentNotNull<object>(null, "object");
17      }
18  }
```

Index

- ./Platform.Exceptions.Tests/EnsuranceTests.cs, 5
- ./Platform.Exceptions.Tests/Ignore/EnsureAlwaysExtensions.cs, 5
- ./Platform.Exceptions.Tests/Ignore/IgnoredEnsuranceTests.cs, 5
- ./Platform.Exceptions/Ensure.cs, 1
- ./Platform.Exceptions/EnsureExtensions.cs, 1
- ./Platform.Exceptions/ExceptionExtensions.cs, 2
- ./Platform.Exceptions/ExtensionRoots/EnsureAlwaysExtensionRoot.cs, 3
- ./Platform.Exceptions/ExtensionRoots/EnsureOnDebugExtensionRoot.cs, 3
- ./Platform.Exceptions/ExtensionRoots/ThrowExtensionRoot.cs, 3
- ./Platform.Exceptions/IgnoredExceptions.cs, 3
- ./Platform.Exceptions/Throw.cs, 4
- ./Platform.Exceptions/ThrowExtensions.cs, 5