

LinksPlatform's Platform.Exceptions Class Library

./Ensure.cs

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

./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
8 namespace Platform.Exceptions
9 {
10    public static class EnsureAlwaysExtensions
11    {
12        #region Always
13
14        [MethodImpl(MethodImplOptions.AggressiveInlining)]
15        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
16        → TArgument argument, string argumentName, string message)
17        where TArgument : class
18        {
19            if (argument == null)
20            {
21                throw new ArgumentNullException(argumentName, message);
22            }
23        }
24
25        [MethodImpl(MethodImplOptions.AggressiveInlining)]
26        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
27        → TArgument argument, string argumentName) where TArgument : class =>
28        → ArgumentNotNull(root, argument, argumentName, $"Argument {argumentName} is null.");
29
30        [MethodImpl(MethodImplOptions.AggressiveInlining)]
31        public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
32        → TArgument argument) where TArgument : class => ArgumentNotNull(root, argument, null);
33
34        [MethodImpl(MethodImplOptions.AggressiveInlining)]
35        public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
36        → Predicate<TArgument> predicate, TArgument argument, string argumentName, string
37        → message)
38        {
39            if (!predicate(argument))
40            {
41                throw new ArgumentException(argumentName, message);
42            }
43        }
44    }
45 }
```

```

37     }
38
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
        Predicate<TArgument> predicate, TArgument argument, string argumentName) =>
        ArgumentMeetsCriteria(root, predicate, argument, argumentName, $"Argument
        {argumentName} is does not meet criteria.");
41
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
        Predicate<TArgument> predicate, TArgument argument) => ArgumentMeetsCriteria(root,
        predicate, argument, null);
44
45     #endregion
46
47     #region OnDebug
48
49     [Conditional("DEBUG")]
50     public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
        TArgument argument, string argumentName, string message) where TArgument : class =>
        Ensure.Always.ArgumentNotNull(argument, argumentName, message);
51
52     [Conditional("DEBUG")]
53     public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
        TArgument argument, string argumentName) where TArgument : class =>
        Ensure.Always.ArgumentNotNull(argument, argumentName);
54
55     [Conditional("DEBUG")]
56     public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
        TArgument argument) where TArgument : class =>
        Ensure.Always.ArgumentNotNull(argument);
57
58     [Conditional("DEBUG")]
59     public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
        root, Predicate<TArgument> predicate, TArgument argument, string argumentName,
        string message) => Ensure.Always.ArgumentMeetsCriteria(predicate, argument,
        argumentName, message);
60
61     [Conditional("DEBUG")]
62     public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
        root, Predicate<TArgument> predicate, TArgument argument, string argumentName) =>
        Ensure.Always.ArgumentMeetsCriteria(predicate, argument, argumentName);
63
64     [Conditional("DEBUG")]
65     public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
        root, Predicate<TArgument> predicate, TArgument argument) =>
        Ensure.Always.ArgumentMeetsCriteria(predicate, argument);
66
67     #endregion
68 }
69 }

```

./ExceptionExtensions.cs

```

1  using System;
2  using System.Text;
3
4  namespace Platform.Exceptions
5  {
6      public static class ExceptionExtensions
7      {
8          public static readonly string ExceptionContentsSeparator = "---";
9          public static readonly string ExceptionStringBuildingFailed = "Unable to format
            exception.";
10
11         public static void Ignore(this Exception exception) =>
            IgnoredExceptions.RaiseExceptionIgnoredEvent(exception);
12
13         public static string ToStringWithAllInnerExceptions(this Exception exception)
14         {
15             try
16             {
17                 var sb = new StringBuilder();
18                 BuildExceptionString(sb, exception, 0);
19                 return sb.ToString();
20             }
21             catch (Exception ex)
22             {
23                 ex.Ignore();
24             }
25         }
26     }
27 }

```

```

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

```

./ExtensionRoots/EnsureAlwaysExtensionRoot.cs

```

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

```

./ExtensionRoots/EnsureOnDebugExtensionRoot.cs

```

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

```

./ExtensionRoots/ThrowExtensionRoot.cs

```

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

```

./IgnoredExceptions.cs

```

1 using System;
2 using System.Collections.Concurrent;
3 using System.Collections.Generic;
4
5 namespace Platform.Exceptions
6 {
7     /// <summary>
8     /// Contains a mechanism for notifying about the occurrence of ignored exceptions, as well
9     ↪ as a mechanism for their collection.
10    /// Содержит механизм уведомления о возникновении игнорируемых исключений, а так же механизм
11    ↪ их сбора.
12    /// </summary>
13    public static class IgnoredExceptions
14    {
15    }
16 }

```

```

13     private static readonly ConcurrentBag<Exception> _exceptionsBag = new
14         ↳ ConcurrentBag<Exception>();
15
16     /// <summary>
17     /// An event that is raised every time an exception has been ignored.
18     /// Событие, которое генерируется каждый раз, когда исключение было проигнорировано.
19     /// </summary>
20     public static event EventHandler<Exception> ExceptionIgnored = OnExceptionIgnored;
21
22     /// <summary>
23     /// Gets an immutable collection with all collected exceptions that were ignored.
24     /// Возвращает неизменяемую коллекцию со всеми собранными исключениями которые были
25     ↳ проигнорированы.
26     /// </summary>
27     public static IReadOnlyCollection<Exception> CollectedExceptions => _exceptionsBag;
28
29     /// <summary>
30     /// Gets or sets a value that determines whether to collect ignored exceptions into
31     ↳ CollectedExceptions.
32     /// Возвращает или устанавливает значение, определяющие нужно ли собирать игнорируемые
33     ↳ исключения в CollectedExceptions.
34     /// </summary>
35     public static bool CollectExceptions { get; set; }
36
37     /// <summary>
38     /// Raises an exception ignored event.
39     /// Генерирует событие игнорирования исключения.
40     /// </summary>
41     /// <param name="exception">The ignored exception. Игнорируемое исключение.</param>
42     /// <remarks>
43     /// It is recommended to call this method in cases where you have a catch block, but you
44     ↳ do not do anything with exception in it.
45     /// Рекомендуется вызывать этот метод в тех случаях, когда у вас есть catch блок, но вы
46     ↳ ничего не делаете в нём с исключением.
47     /// </remarks>
48     public static void RaiseExceptionIgnoredEvent(Exception exception) =>
49         ↳ ExceptionIgnored.Invoke(null, exception);
50
51     private static void OnExceptionIgnored(object sender, Exception exception)
52     {
53         if (CollectExceptions)
54         {
55             _exceptionsBag.Add(exception);
56         }
57     }
58 }

```

./Throw.cs

```

1 using Platform.Exceptions.ExtensionRoots;
2
3 namespace Platform.Exceptions
4 {
5     /// <summary>
6     /// Contains an instance of an extensible class that can be supplemented with static helper
7     ↳ methods by using the extension mechanism. These methods throw exceptions.
8     /// Содержит экземпляр расширяемого класса, который можно дополнять статическими
9     ↳ вспомогательными методами путём использования механизма расширений. Эти методы
10    ↳ занимают выбрасыванием исключений.
11    /// </summary>
12    public static class Throw
13    {
14        /// <summary>
15        /// Gets an instance of the extension root class that contains helper methods for
16        ↳ throwing exceptions.
17        /// Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
18        ↳ методы для выбрасывания исключений.
19        /// </summary>
20        public static readonly ThrowExtensionRoot A = new ThrowExtensionRoot();
21    }
22 }

```

./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
7 namespace Platform.Exceptions
8 {
9     public static class ThrowExtensions
10    {
11        [MethodImpl(MethodImplOptions.AggressiveInlining)]
12        public static void NotSupportedException(this ThrowExtensionRoot root) => throw new
            ↳ NotSupportedException();
13
14        [MethodImpl(MethodImplOptions.AggressiveInlining)]
15        public static T NotSupportedExceptionAndReturn<T>(this ThrowExtensionRoot root) => throw
            ↳ new NotSupportedException();
16
17        [MethodImpl(MethodImplOptions.AggressiveInlining)]
18        public static void NotImplementedException(this ThrowExtensionRoot root) => throw new
            ↳ NotImplementedException();
19
20        [MethodImpl(MethodImplOptions.AggressiveInlining)]
21        public static T NotImplementedExceptionAndReturn<T>(this ThrowExtensionRoot root) =>
            ↳ throw new NotImplementedException();
22    }
23 }
```

Index

- ./Ensure.cs, 1
- ./EnsureExtensions.cs, 1
- ./ExceptionExtensions.cs, 2
- ./ExtensionRoots/EnsureAlwaysExtensionRoot.cs, 3
- ./ExtensionRoots/EnsureOnDebugExtensionRoot.cs, 3
- ./ExtensionRoots/ThrowExtensionRoot.cs, 3
- ./IgnoredExceptions.cs, 3
- ./Throw.cs, 4
- ./ThrowExtensions.cs, 4