

LinksPlatform's Platform.Exceptions Class Library

1.1 ./csharp/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 }

```

1.2 ./csharp/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
8 namespace Platform.Exceptions
9 {
10     /// <summary>
11     /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/>
12     ///   ↳ and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
13     /// <para>Предоставляет набор методов расширения для объектов <see
14     ///   ↳ cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
15     /// </summary>
16     public static class EnsureExtensions
17     {
18         #region Always
19
20         /// <summary>
21         /// <para>Ensures that argument is not null. This check is performed regardless of the
22         ///   ↳ build configuration.</para>
23         /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется независимо
24         ///   ↳ от конфигурации сборки.</para>
25         /// </summary>
26         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
27         ///   ↳ аргумента.</para></typeparam>
28         /// <param name="root"><para>The extension root to which this method is
29         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
30         /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
31         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
32         ///   ↳ аргумента.</para></param>
33         /// <param name="message"><para>The message of the thrown
34         ///   ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
37             ↳ TArgument argument, string argumentName, string message)
38             where TArgument : class
39     }
40 }

```

```

30 {
31     if (argument == null)
32     {
33         throw new ArgumentNullException(argumentName, message);
34     }
35 }
36
37 /// <summary>
38 /// <para>Ensures that argument is not null. This check is performed regardless of the
39   → build configuration.</para>
40 /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется независимо
41   → от конфигурации сборки.</para>
42 /// </summary>
43 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
44   → аргумента.</para></typeparam>
45 /// <param name="root"><para>The extension root to which this method is
46   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
47 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
48 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
49   → аргумента.</para></param>
50 [MethodImpl(MethodImplOptions.AggressiveInlining)]
51 public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
52   → TArgument argument, string argumentName) where TArgument : class =>
53   → ArgumentNotNull(root, argument, argumentName, $"Argument {argumentName} is null.");
54
55 /// <summary>
56 /// <para>Ensures that argument is not null. This check is performed regardless of the
57   → build configuration.</para>
58 /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется независимо
59   → от конфигурации сборки.</para>
60 /// </summary>
61 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
62   → аргумента.</para></typeparam>
63 /// <param name="root"><para>The extension root to which this method is
64   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
65 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
66 [MethodImpl(MethodImplOptions.AggressiveInlining)]
67 public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
68   → TArgument argument) where TArgument : class => ArgumentNotNull(root, argument, null);
69
70 /// <summary>
71 /// <para>Ensures that the argument meets the criteria. This check is performed
72   → regardless of the build configuration.</para>
73 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
74   → независимо от конфигурации сборки.</para>
75 /// </summary>
76 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
77   → аргумента.</para></typeparam>
78 /// <param name="root"><para>The extension root to which this method is
79   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
80 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
81 /// <param name="predicate"><para>A predicate that determines whether the argument meets
82   → a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
83   → критерию.</para></param>
84 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
85   → аргумента.</para></param>
86 /// <param name="message"><para>The message of the thrown
87   → exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
88 [MethodImpl(MethodImplOptions.AggressiveInlining)]
89 public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
90   → TArgument argument, Predicate<TArgument> predicate, string argumentName, string
91   → message)
92 {
93     if (!predicate(argument))
94     {
95         throw new ArgumentException(message, argumentName);
96     }
97 }
98
99 /// <summary>
100 /// <para>Ensures that the argument meets the criteria. This check is performed
101   → regardless of the build configuration.</para>
102 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
103   → независимо от конфигурации сборки.</para>
104 /// </summary>

```

```

81  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    → аргумента.</para></typeparam>
82  /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
83  /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
84  /// <param name="predicate"><para>A predicate that determines whether the argument meets
    → a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
    → критерию.</para></param>
85  /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    → аргумента.</para></param>
86  [MethodImpl(MethodImplOptions.AggressiveInlining)]
87  public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
    → TArgument argument, Predicate<TArgument> predicate, string argumentName) =>
    → ArgumentMeetsCriteria(root, argument, predicate, argumentName, $"Argument
    → {argumentName} does not meet the criteria.");
88
89  /// <summary>
90  /// <para>Ensures that the argument meets the criteria. This check is performed
    → regardless of the build configuration.</para>
91  /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
    → независимо от конфигурации сборки.</para>
92  /// </summary>
93  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    → аргумента.</para></typeparam>
94  /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
95  /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
96  /// <param name="predicate"><para>A predicate that determines whether the argument meets
    → a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
    → критерию.</para></param>
97  [MethodImpl(MethodImplOptions.AggressiveInlining)]
98  public static void ArgumentMeetsCriteria<TArgument>(this EnsureAlwaysExtensionRoot root,
    → TArgument argument, Predicate<TArgument> predicate) => ArgumentMeetsCriteria(root,
    → argument, predicate, null);
99
100 #endregion
101
102 #region OnDebug
103
104  /// <summary>
105  /// <para>Ensures that argument is not null. This check is performed only for DEBUG
    → build configuration.</para>
106  /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
    → конфигурации сборки DEBUG.</para>
107  /// </summary>
108  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    → аргумента.</para></typeparam>
109  /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
110  /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
111  /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    → аргумента.</para></param>
112  /// <param name="message"><para>The message of the thrown
    → exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
113  [Conditional("DEBUG")]
114  public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
    → TArgument argument, string argumentName, string message) where TArgument : class =>
    → Ensure.Always.ArgumentNotNull(argument, argumentName, message);
115
116  /// <summary>
117  /// <para>Ensures that argument is not null. This check is performed only for DEBUG
    → build configuration.</para>
118  /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
    → конфигурации сборки DEBUG.</para>
119  /// </summary>
120  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    → аргумента.</para></typeparam>
121  /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
122  /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
123  /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    → аргумента.</para></param>
124  [Conditional("DEBUG")]
125  public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
    → TArgument argument, string argumentName) where TArgument : class =>
    → Ensure.Always.ArgumentNotNull(argument, argumentName);

```

```

126
127 /// <summary>
128 /// <para>Ensures that argument is not null. This check is performed only for DEBUG
    ↳ build configuration.</para>
129 /// <para>Гарантирует, что аргумент не нулевой. Эта проверка выполняется только для
    ↳ конфигурации сборки DEBUG.</para>
130 /// </summary>
131 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
132 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
133 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
134 [Conditional("DEBUG")]
135 public static void ArgumentNotNull<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argument) where TArgument : class =>
    ↳ Ensure.Always.ArgumentNotNull(argument);
136
137 /// <summary>
138 /// <para>Ensures that the argument meets the criteria. This check is performed only for
    ↳ DEBUG build configuration.</para>
139 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
    ↳ только для конфигурации сборки DEBUG.</para>
140 /// </summary>
141 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
142 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
143 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
144 /// <param name="predicate"><para>A predicate that determines whether the argument meets
    ↳ a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
    ↳ критерию.</para></param>
145 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
146 /// <param name="message"><para>The message of the thrown
    ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
147 [Conditional("DEBUG")]
148 public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
    ↳ root, TArgument argument, Predicate<TArgument> predicate, string argumentName,
    ↳ string message) => Ensure.Always.ArgumentMeetsCriteria(argument, predicate,
    ↳ argumentName, message);
149
150 /// <summary>
151 /// <para>Ensures that the argument meets the criteria. This check is performed only for
    ↳ DEBUG build configuration.</para>
152 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
    ↳ только для конфигурации сборки DEBUG.</para>
153 /// </summary>
154 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
155 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
156 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
157 /// <param name="predicate"><para>A predicate that determines whether the argument meets
    ↳ a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
    ↳ критерию.</para></param>
158 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
159 [Conditional("DEBUG")]
160 public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
    ↳ root, TArgument argument, Predicate<TArgument> predicate, string argumentName) =>
    ↳ Ensure.Always.ArgumentMeetsCriteria(argument, predicate, argumentName);
161
162 /// <summary>
163 /// <para>Ensures that the argument meets the criteria. This check is performed only for
    ↳ DEBUG build configuration.</para>
164 /// <para>Гарантирует, что аргумент соответствует критерию. Эта проверка выполняется
    ↳ только для конфигурации сборки DEBUG.</para>
165 /// </summary>
166 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
167 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
168 /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>

```

```

169     /// <param name="predicate"><para>A predicate that determines whether the argument meets
    ↪ a criterion.</para><para>Предикат определяющий, соответствует ли аргумент
    ↪ критерию.</para></param>
170 [Conditional("DEBUG")]
171 public static void ArgumentMeetsCriteria<TArgument>(this EnsureOnDebugExtensionRoot
    ↪ root, TArgument argument, Predicate<TArgument> predicate) =>
    ↪ Ensure.Always.ArgumentMeetsCriteria(argument, predicate);

172 #endregion
173 }
174 }
175 }

```

1.3 ./csharp/Platform.Exceptions/ExceptionExtensions.cs

```

1 using System;
2 using System.Text;
3
4 namespace Platform.Exceptions
5 {
6     /// <summary>
7     /// <para>Provides a set of extension methods for <see cref="Exception"/> objects.</para>
8     /// <para>Предоставляет набор методов расширения для объектов <see cref="Exception"/>.</para>
9     /// </summary>
10    public static class ExceptionExtensions
11    {
12        /// <summary>
13        /// <para>Represents the separator used within the process of generating a
    ↪ representation string (<see cref="ToStringWithAllInnerExceptions(Exception)"/>) to
    ↪ separate different inner exceptions from each other. This field is constant.</para>
14        /// <para>Представляет разделитель, используемый внутри процесса формирования
    ↪ строки-представления (<see cref="ToStringWithAllInnerExceptions(Exception)"/>) для
    ↪ разделения различных внутренних исключений друг от друга. Это поле является
    ↪ константой.</para>
15        /// </summary>
16        public static readonly string ExceptionContentsSeparator = "---";
17
18        /// <summary>
19        /// <para>Represents a string returned from <see
    ↪ cref="ToStringWithAllInnerExceptions(Exception)"/> in the event of an unsuccessful
    ↪ attempt to format an exception. This field is a constant.</para>
20        /// <para>Представляет строку выдаваемую из <see
    ↪ cref="ToStringWithAllInnerExceptions(Exception)"/> в случае неудачной попытки
    ↪ форматирования исключения. Это поле является константой.</para>
21        /// </summary>
22        public static readonly string ExceptionStringBuildingFailed = "Unable to format
    ↪ exception.";
23
24        /// <summary>
25        /// <para>Ignores the exception, notifying the <see cref = "IgnoredExceptions" /> class
    ↪ about it.</para>
26        /// <para>Игнорирует исключение, уведомляя об этом класс <see
    ↪ cref="IgnoredExceptions"/>.</para>
27        /// </summary>
28        /// <param name="exception"><para></para><para></para></param>
29        public static void Ignore(this Exception exception) =>
    ↪ IgnoredExceptions.RaiseExceptionIgnoredEvent(exception);
30
31        /// <summary>
32        /// <para>Returns a string that represents the specified exception with all its inner
    ↪ exceptions.</para>
33        /// <para>Возвращает строку, которая представляет указанное исключение со всеми его
    ↪ внутренними исключениями.</para>
34        /// </summary>
35        /// <param name="exception"><para>The exception that will be represented as a
    ↪ string.</para><para>Исключение, которое будет представлено в виде
    ↪ строки.</para></param>
36        /// <returns><para>A string that represents the specified exception with all its inner
    ↪ exceptions.</para><para>Строку, которая представляет указанное исключение со всеми
    ↪ его внутренними исключениями.</para></returns>
37        public static string ToStringWithAllInnerExceptions(this Exception exception)
38        {
39            try
40            {
41                var sb = new StringBuilder();
42                sb.BuildExceptionString(exception, 0);
43                return sb.ToString();
44            }
45            catch (Exception ex)

```

```

46     {
47         ex.Ignore();
48         return ExceptionStringBuilderFailed;
49     }
50 }
51
52 /// <summary>
53 /// <para>
54 /// Builds the exception string using the specified sb.
55 /// </para>
56 /// <para></para>
57 /// </summary>
58 /// <param name="sb">
59 /// <para>The sb.</para>
60 /// <para></para>
61 /// </param>
62 /// <param name="exception">
63 /// <para>The exception.</para>
64 /// <para></para>
65 /// </param>
66 /// <param name="level">
67 /// <para>The level.</para>
68 /// <para></para>
69 /// </param>
70 private static void BuildExceptionString(this StringBuilder sb, Exception exception, int
    ↪ level)
71 {
72     sb.Indent(level);
73     sb.AppendLine(exception.Message);
74     sb.Indent(level);
75     sb.AppendLine(ExceptionContentsSeparator);
76     if (exception.InnerException != null)
77     {
78         sb.Indent(level);
79         sb.AppendLine("Inner exception: ");
80         sb.BuildExceptionString(exception.InnerException, level + 1);
81     }
82     sb.Indent(level);
83     sb.AppendLine(ExceptionContentsSeparator);
84     sb.Indent(level);
85     sb.AppendLine(exception.StackTrace);
86 }
87
88 /// <summary>
89 /// <para>
90 /// Indents the sb.
91 /// </para>
92 /// <para></para>
93 /// </summary>
94 /// <param name="sb">
95 /// <para>The sb.</para>
96 /// <para></para>
97 /// </param>
98 /// <param name="level">
99 /// <para>The level.</para>
100 /// <para></para>
101 /// </param>
102 private static void Indent(this StringBuilder sb, int level) => sb.Append('\t', level);
103 }
104 }

```

1.4 ./csharp/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 }

```

1.5 ./csharp/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 }

```

1.6 ./csharp/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 }

```

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

```

```

48     public static void RaiseExceptionIgnoredEvent(Exception exception) =>
49         ↳ ExceptionIgnored.Invoke(null, exception);
50
51     /// <summary>
52     /// <para>
53     /// Ons the exception ignored using the specified sender.
54     /// </para>
55     /// </summary>
56     /// <param name="sender">
57     /// <para>The sender.</para>
58     /// </param>
59     /// <param name="exception">
60     /// <para>The exception.</para>
61     /// </param>
62     private static void OnExceptionIgnored(object sender, Exception exception)
63     {
64         if (CollectExceptions)
65         {
66             _exceptionsBag.Add(exception);
67         }
68     }
69 }
70
71 }
72 }

```

1.8 ./csharp/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
7      ↳ helper methods by using the extension mechanism. These methods throw exceptions.</para>
8      /// <para>Содержит экземпляр расширяемого класса, который можно дополнять статическими
9      ↳ вспомогательными методами путём использования механизма расширений. Эти методы
10     ↳ занимаются выбрасыванием исключений.</para>
11     /// </summary>
12     public static class Throw
13     {
14         /// <summary>
15         /// <para>Gets an instance of the extension root class that contains helper methods for
16         ↳ throwing exceptions.</para>
17         /// <para>Возвращает экземпляр класса корня-расширения, который содержит вспомогательные
18         ↳ методы для выбрасывания исключений.</para>
19         /// </summary>
20         public static readonly ThrowExtensionRoot A = new ThrowExtensionRoot();
21     }
22 }

```

1.9 ./csharp/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
7  namespace Platform.Exceptions
8  {
9      /// <summary>
10     /// <para>Provides a set of extension methods for <see cref="ThrowExtensionRoot"/>
11     ↳ objects.</para>
12     /// <para>Предоставляет набор методов расширения для объектов <see
13     ↳ cref="ThrowExtensionRoot"/>.</para>
14     /// </summary>
15     public static class ThrowExtensions
16     {
17         /// <summary>
18         /// <para>Throws a new <see cref="System.NotSupportedException"/>.</para>
19         /// <para>Выбрасывает новое <see cref="System.NotSupportedException"/>.</para>
20         /// </summary>
21         /// <param name="root"><para>The extension root to which this method is
22         ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static void NotSupportedException(this ThrowExtensionRoot root) => throw new
25         ↳ NotSupportedException();
26     }
27 }

```



```

23     /// <summary>
24     /// <para>Throws a new <see cref="System.NotSupportedException"/>, while returning a
    → value of <typeparamref name="TReturn"/> type.</para>
25     /// <para>Выбрасывает новое <see cref="System.NotSupportedException"/>, возвращая при
    → этом значение типа <typeparamref name="TReturn"/>.</para>
26     /// </summary>
27     /// <typeparam name="TReturn"><para>The type of returned value.</para><para>Тип
    → возвращаемого значения.</para></typeparam>
28     /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
29     /// <returns><para>A value of <typeparamref name="TReturn"/> type.</para><para>Значение
    → типа <typeparamref name="TReturn"/>.</para></returns>
30     [MethodImpl(MethodImplOptions.AggressiveInlining)]
31     public static TReturn NotSupportedExceptionAndReturn<TReturn>(this ThrowExtensionRoot
    → root) => throw new NotSupportedException();
32
33     /// <summary>
34     /// <para>Throws a new <see cref="System.NotImplementedException"/>.</para>
35     /// <para>Выбрасывает новое <see cref="System.NotImplementedException"/>.</para>
36     /// </summary>
37     /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public static void NotImplementedException(this ThrowExtensionRoot root) => throw new
    → NotImplementedException();
40
41     /// <summary>
42     /// <para>Throws a new <see cref="System.NotImplementedException"/>, while returning a
    → value of <typeparamref name="TReturn"/> type.</para>
43     /// <para>Выбрасывает новое <see cref="System.NotImplementedException"/>, возвращая при
    → этом значение типа <typeparamref name="TReturn"/>.</para>
44     /// </summary>
45     /// <typeparam name="TReturn"><para>The type of returned value.</para><para>Тип
    → возвращаемого значения.</para></typeparam>
46     /// <param name="root"><para>The extension root to which this method is
    → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
47     /// <returns><para>A value of <typeparamref name="TReturn"/> type.</para><para>Значение
    → типа <typeparamref name="TReturn"/>.</para></returns>
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     public static TReturn NotImplementedExceptionAndReturn<TReturn>(this ThrowExtensionRoot
    → root) => throw new NotImplementedException();
50 }
51 }

```

1.10 ./csharp/Platform.Exceptions.Tests/EnsuranceTests.cs

```

1  using System;
2  using Xunit;
3
4  namespace Platform.Exceptions.Tests
5  {
6      /// <summary>
7      /// <para>
8      /// Represents the ensurance tests.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public static class EnsuranceTests
13     {
14         /// <summary>
15         /// <para>
16         /// Tests that argument not null ensurance test.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         [Fact]
21         public static void ArgumentNotNullEnsuranceTest()
22         {
23             // Should throw an exception (even if in neighbour "Ignore" namespace it was
    → overridden, but here this namespace is not used)
24             Assert.Throws<ArgumentNullException>(() =>
    → Ensure.Always.ArgumentNotNull<object>(null, "object"));
25         }
26     }
27 }

```

1.11 ./csharp/Platform.Exceptions.Tests/Ignore/EnsureExtensions.cs

```

1  using System.Diagnostics;
2  using Platform.Exceptions.ExtensionRoots;
3
4  namespace Platform.Exceptions.Tests.Ignore
5  {
6      /// <summary>
7      /// <para>
8      /// Represents the ensure extensions.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public static class EnsureExtensions
13     {
14         /// <summary>
15         /// <para>
16         /// Arguments the not null using the specified root.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         /// <typeparam name="TArgument">
21         /// <para>The argument.</para>
22         /// <para></para>
23         /// </typeparam>
24         /// <param name="root">
25         /// <para>The root.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="argument">
29         /// <para>The argument.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="argumentName">
33         /// <para>The argument name.</para>
34         /// <para></para>
35         /// </param>
36         [Conditional("DEBUG")]
37         public static void ArgumentNotNull<TArgument>(this EnsureAlwaysExtensionRoot root,
38             ↪ TArgument argument, string argumentName)
39             where TArgument : class
40         {
41             // Override logic to do nothing (this should be used to reduce the overhead of the
42             ↪ Ensure checks, when it is critical to performance)
43         }
44     }
45 }

```

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

```

1  using Xunit;
2
3  namespace Platform.Exceptions.Tests.Ignore
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the ignored ensurance tests.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public static class IgnoredEnsuranceTests
12     {
13         /// <summary>
14         /// <para>
15         /// Tests that ensurance ignored test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void EnsuranceIgnoredTest()
21         {
22             // Should not throw an exception (because logic is overridden in
23             ↪ EnsureAlwaysExtensions that is located within the same namespace)
24             // And even should be optimized out at RELEASE (because method is now marked
25             ↪ conditional DEBUG)
26             // This can be useful in performance critical situations there even an check for
27             ↪ exception is hurting performance enough
28             Ensure.Always.ArgumentNotNull<object>(null, "object");
29         }
30     }
31 }

```


Index

- ./csharp/Platform.Exceptions.Tests/EnsuranceTests.cs, 9
- ./csharp/Platform.Exceptions.Tests/Ignore/EnsureExtensions.cs, 9
- ./csharp/Platform.Exceptions.Tests/Ignore/IgnoredEnsuranceTests.cs, 10
- ./csharp/Platform.Exceptions/Ensure.cs, 1
- ./csharp/Platform.Exceptions/EnsureExtensions.cs, 1
- ./csharp/Platform.Exceptions/ExceptionExtensions.cs, 5
- ./csharp/Platform.Exceptions/ExtensionRoots/EnsureAlwaysExtensionRoot.cs, 6
- ./csharp/Platform.Exceptions/ExtensionRoots/EnsureOnDebugExtensionRoot.cs, 6
- ./csharp/Platform.Exceptions/ExtensionRoots/ThrowExtensionRoot.cs, 7
- ./csharp/Platform.Exceptions/IgnoredExceptions.cs, 7
- ./csharp/Platform.Exceptions/Throw.cs, 8
- ./csharp/Platform.Exceptions/ThrowExtensions.cs, 8