

LinksPlatform's Platform.Ranges Class Library

./EnsureExtensions.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Runtime.CompilerServices;
5  using Platform.Exceptions;
6  using Platform.Exceptions.ExtensionRoots;
7
8  #pragma warning disable IDE0060 // Remove unused parameter
9
10 namespace Platform.Ranges
11 {
12     /// <summary>
13     /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/>
14     ///   ↳ and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
15     /// <para>Предоставляет набор методов расширения для объектов <see
16     ///   ↳ cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
17     /// </summary>
18     public static class EnsureExtensions
19     {
20         private const string DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage = "Maximum
21         ↳ should be greater or equal to minimum.";
22
23         #region Always
24
25         /// <summary>
26         /// <para>Ensures that the argument with the maximum value is greater than or equal to
27         ///   ↳ the minimum value. This check is performed regardless of the build
28         ///   ↳ configuration.</para>
29         /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
30         ///   ↳ минимальному значению. Эта проверка выполняется независимо от конфигурации
31         ///   ↳ сборки.</para>
32         /// </summary>
33         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
34         ///   ↳ аргумента.</para></typeparam>
35         /// <param name="root"><para>The extension root to which this method is
36         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
37         /// <param name="minimumArgument"><para>The argument with the minimum
38         ///   ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
39         /// <param name="maximumArgument"><para>The argument with the maximum
40         ///   ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
41         /// <param name="maximumArgumentName"><para>The name of argument with the maximum
42         ///   ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>
43         /// <param name="messageBuilder"><para>The thrown exception's message building <see
44         ///   ↳ cref="Func{String}"/>.</para><para>Собирающая сообщение для выбрасываемого
45         ///   ↳ исключения <see cref="Func{String}"/>.</para></param>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
48         ↳ EnsureAlwaysExtensionRoot root, TArgument minimumArgument, TArgument
49         ↳ maximumArgument, string maximumArgumentName, Func<string> messageBuilder)
50         {
51             if (Comparer<TArgument>.Default.Compare(maximumArgument, minimumArgument) < 0)
52             {
53                 throw new ArgumentException(messageBuilder(), maximumArgumentName);
54             }
55         }
56
57         /// <summary>
58         /// <para>Ensures that the argument with the maximum value is greater than or equal to
59         ///   ↳ the minimum value. This check is performed regardless of the build
60         ///   ↳ configuration.</para>
61         /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
62         ///   ↳ минимальному значению. Эта проверка выполняется независимо от конфигурации
63         ///   ↳ сборки.</para>
64         /// </summary>
65         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
66         ///   ↳ аргумента.</para></typeparam>
67         /// <param name="root"><para>The extension root to which this method is
68         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
69         /// <param name="minimumArgument"><para>The argument with the minimum
70         ///   ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
71         /// <param name="maximumArgument"><para>The argument with the maximum
72         ///   ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
73         /// <param name="maximumArgumentName"><para>The name of argument with the maximum
74         ///   ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>

```

```

50  /// <param name="message"><para>The message of the thrown
    ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
51  [MethodImpl(MethodImplOptions.AggressiveInlining)]
52  public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureAlwaysExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument, string maximumArgumentName, string message)
53  {
54      string messageBuilder() => message;
55      MaximumArgumentIsGreaterOrEqualToMinimum(root, minimumArgument, maximumArgument,
    ↳ maximumArgumentName, messageBuilder);
56  }
57
58  /// <summary>
59  /// <para>Ensures that the argument with the maximum value is greater than or equal to
    ↳ the minimum value. This check is performed regardless of the build
    ↳ configuration.</para>
60  /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
    ↳ минимальному значению. Эта проверка выполняется независимо от конфигурации
    ↳ сборки.</para>
61  /// </summary>
62  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
63  /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
64  /// <param name="minimumArgument"><para>The argument with the minimum
    ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
65  /// <param name="maximumArgument"><para>The argument with the maximum
    ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
66  /// <param name="maximumArgumentName"><para>The name of argument with the maximum
    ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>
67  [MethodImpl(MethodImplOptions.AggressiveInlining)]
68  public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureAlwaysExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument, string maximumArgumentName) =>
    ↳ MaximumArgumentIsGreaterOrEqualToMinimum(root, minimumArgument, maximumArgument,
    ↳ nameof(maximumArgument), DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage);
69
70  /// <summary>
71  /// <para>Ensures that the argument with the maximum value is greater than or equal to
    ↳ the minimum value. This check is performed regardless of the build
    ↳ configuration.</para>
72  /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
    ↳ минимальному значению. Эта проверка выполняется независимо от конфигурации
    ↳ сборки.</para>
73  /// </summary>
74  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
75  /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
76  /// <param name="minimumArgument"><para>The argument with the minimum
    ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
77  /// <param name="maximumArgument"><para>The argument with the maximum
    ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
78  [MethodImpl(MethodImplOptions.AggressiveInlining)]
79  public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureAlwaysExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument) => MaximumArgumentIsGreaterOrEqualToMinimum(root, minimumArgument,
    ↳ maximumArgument, nameof(maximumArgument));
80
81  /// <summary>
82  /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed regardless of the build configuration.</para>
83  /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется независимо от конфигурации сборки.</para>
84  /// </summary>
85  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
86  /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
87  /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
    ↳ аргумента.</para></param>
88  /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↳ диапазона.</para></param>
89  /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>

```

```

90  /// <param name="messageBuilder"><para>The thrown exception's message building <see
    ↪ cref="Func{String}" />.</para><para>Собирающая сообщение для выбрасываемого
    ↪ исключения <see cref="Func{String}" />.</para></param>
91  [MethodImpl(MethodImplOptions.AggressiveInlining)]
92  public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
    ↪ TArgument argumentValue, Range<TArgument> range, string argumentName, Func<string>
    ↪ messageBuilder)
93  {
94      if (!range.ContainsValue(argumentValue))
95      {
96          throw new ArgumentOutOfRangeException(argumentName, argumentValue,
    ↪ messageBuilder());
97      }
98  }
99
100  /// <summary>
101  /// <para>Ensures that the argument value is in the specified range. This check is
    ↪ performed regardless of the build configuration.</para>
102  /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↪ проверка выполняется независимо от конфигурации сборки.</para>
103  /// </summary>
104  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↪ аргумента.</para></typeparam>
105  /// <param name="root"><para>The extension root to which this method is
    ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
106  /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
    ↪ аргумента.</para></param>
107  /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↪ диапазона.</para></param>
108  /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↪ аргумента.</para></param>
109  /// <param name="message"><para>The message of the thrown
    ↪ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
110  [MethodImpl(MethodImplOptions.AggressiveInlining)]
111  public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
    ↪ TArgument argumentValue, Range<TArgument> range, string argumentName, string message)
112  {
113      string messageBuilder() => message;
114      ArgumentInRange(root, argumentValue, range, argumentName, messageBuilder);
115  }
116
117  /// <summary>
118  /// <para>Ensures that the argument value is in the specified range. This check is
    ↪ performed regardless of the build configuration.</para>
119  /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↪ проверка выполняется независимо от конфигурации сборки.</para>
120  /// </summary>
121  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↪ аргумента.</para></typeparam>
122  /// <param name="root"><para>The extension root to which this method is
    ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
123  /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
    ↪ аргумента.</para></param>
124  /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↪ диапазона.</para></param>
125  /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↪ аргумента.</para></param>
126  [MethodImpl(MethodImplOptions.AggressiveInlining)]
127  public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
    ↪ TArgument argumentValue, Range<TArgument> range, string argumentName)
128  {
129      string messageBuilder() => $"Argument value [{argumentValue}] is out of range
    ↪ {range}.";
130      ArgumentInRange(root, argumentValue, range, argumentName, messageBuilder);
131  }
132
133  /// <summary>
134  /// <para>Ensures that the argument value is in the specified range. This check is
    ↪ performed regardless of the build configuration.</para>
135  /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↪ проверка выполняется независимо от конфигурации сборки.</para>
136  /// </summary>
137  /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↪ аргумента.</para></typeparam>
138  /// <param name="root"><para>The extension root to which this method is
    ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>

```

```

139 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
140   → аргумента.</para></param>
141 /// <param name="minimum"><para>The minimum possible argument's
142   → value.</para><para>Минимально возможное значение аргумента.</para></param>
143 /// <param name="maximum"><para>The maximum possible argument's
144   → value.</para><para>Максимально возможное значение аргумента.</para></param>
145 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
146   → аргумента.</para></param>
147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
149   → TArgument argumentValue, TArgument minimum, TArgument maximum, string argumentName)
150   → => ArgumentInRange(root, argumentValue, new Range<TArgument>(minimum, maximum),
151   → argumentName);
152
153 /// <summary>
154 /// <para>Ensures that the argument value is in the specified range. This check is
155   → performed regardless of the build configuration.</para>
156 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
157   → проверка выполняется независимо от конфигурации сборки.</para>
158 /// </summary>
159 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
160   → аргумента.</para></typeparam>
161 /// <param name="root"><para>The extension root to which this method is
162   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
163 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
164   → аргумента.</para></param>
165 /// <param name="minimum"><para>The minimum possible argument's
166   → value.</para><para>Минимально возможное значение аргумента.</para></param>
167 /// <param name="maximum"><para>The maximum possible argument's
168   → value.</para><para>Максимально возможное значение аргумента.</para></param>
169 [MethodImpl(MethodImplOptions.AggressiveInlining)]
170 public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
171   → TArgument argumentValue, TArgument minimum, TArgument maximum) =>
172   → ArgumentInRange(root, argumentValue, new Range<TArgument>(minimum, maximum), null);
173
174 /// <summary>
175 /// <para>Ensures that the argument value is in the specified range. This check is
176   → performed regardless of the build configuration.</para>
177 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
178   → проверка выполняется независимо от конфигурации сборки.</para>
179 /// </summary>
180 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
181   → аргумента.</para></typeparam>
182 /// <param name="root"><para>The extension root to which this method is
183   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
184 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
185   → аргумента.</para></param>
186 /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
187   → диапазона.</para></param>
188 [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
190   → TArgument argumentValue, Range<TArgument> range) => ArgumentInRange(root,
191   → argumentValue, range, null);
192
193 #endregion
194
195 #region OnDebug
196
197 /// <summary>
198 /// <para>Ensures that the argument with the maximum value is greater than or equal to
199   → the minimum value. This check is performed only for DEBUG build configuration.</para>
200 /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
201   → минимальному значению. Эта проверка выполняется только для конфигурации сборки
202   → DEBUG.</para>
203 /// </summary>
204 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
205   → аргумента.</para></typeparam>
206 /// <param name="root"><para>The extension root to which this method is
207   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
208 /// <param name="minimumArgument"><para>The argument with the minimum
209   → value.</para><para>Аргумент с минимальным значением.</para></param>
209 /// <param name="maximumArgument"><para>The argument with the maximum
210   → value.</para><para>Аргумент с максимальным значением.</para></param>
211 /// <param name="maximumArgumentName"><para>The name of argument with the maximum
212   → value.</para><para>Имя аргумента с максимальным значением.</para></param>

```

```

182 /// <param name="messageBuilder"><para>The thrown exception's message building <see
    ↳ cref="Func{String}" />.</para><para>Собирающая сообщения для выбрасываемого
    ↳ исключения <see cref="Func{String}" />.</para></param>
183 [Conditional("DEBUG")]
184 public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureOnDebugExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument, string maximumArgumentName, Func<string> messageBuilder) =>
    ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimumArgument,
    ↳ maximumArgument, maximumArgumentName, messageBuilder);
185
186 /// <summary>
187 /// <para>Ensures that the argument with the maximum value is greater than or equal to
    ↳ the minimum value. This check is performed only for DEBUG build configuration.</para>
188 /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
    ↳ минимальному значению. Эта проверка выполняется только для конфигурации сборки
    ↳ DEBUG.</para>
189 /// </summary>
190 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
191 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
192 /// <param name="minimumArgument"><para>The argument with the minimum
    ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
193 /// <param name="maximumArgument"><para>The argument with the maximum
    ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
194 /// <param name="maximumArgumentName"><para>The name of argument with the maximum
    ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>
195 /// <param name="message"><para>The message of the thrown
    ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
196 [Conditional("DEBUG")]
197 public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureOnDebugExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument, string maximumArgumentName, string message) =>
    ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimumArgument,
    ↳ maximumArgument, maximumArgumentName, message);
198
199 /// <summary>
200 /// <para>Ensures that the argument with the maximum value is greater than or equal to
    ↳ the minimum value. This check is performed only for DEBUG build configuration.</para>
201 /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
    ↳ минимальному значению. Эта проверка выполняется только для конфигурации сборки
    ↳ DEBUG.</para>
202 /// </summary>
203 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
204 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
205 /// <param name="minimumArgument"><para>The argument with the minimum
    ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
206 /// <param name="maximumArgument"><para>The argument with the maximum
    ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
207 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
208 [Conditional("DEBUG")]
209 public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureOnDebugExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument, string argumentName) =>
    ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimumArgument,
    ↳ maximumArgument, argumentName);
210
211 /// <summary>
212 /// <para>Ensures that the argument with the maximum value is greater than or equal to
    ↳ the minimum value. This check is performed only for DEBUG build configuration.</para>
213 /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
    ↳ минимальному значению. Эта проверка выполняется только для конфигурации сборки
    ↳ DEBUG.</para>
214 /// </summary>
215 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
216 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
217 /// <param name="minimumArgument"><para>The argument with the minimum
    ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
218 /// <param name="maximumArgument"><para>The argument with the maximum
    ↳ value.</para><para>Аргумент с максимальным значением.</para></param>

```

```

219 [Conditional("DEBUG")]
220 public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
    ↳ EnsureOnDebugExtensionRoot root, TArgument minimumArgument, TArgument
    ↳ maximumArgument) =>
    ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimumArgument,
    ↳ maximumArgument, null);
221
222 /// <summary>
223 /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed only for DEBUG build configuration.</para>
224 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется только для конфигурации сборки DEBUG.</para>
225 /// </summary>
226 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
227 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
228 /// <param name="argument"></param>
229 /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↳ диапазона.</para></param>
230 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
231 /// <param name="messageBuilder"><para>The thrown exception's message building <see
    ↳ cref="Func{String}"/>.</para><para>Собирающая сообщение для выбрасываемого
    ↳ исключения <see cref="Func{String}"/>.</para></param>
232 [Conditional("DEBUG")]
233 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argument, Range<TArgument> range, string argumentName, Func<string>
    ↳ messageBuilder) => Ensure.Always.ArgumentInRange(argument, range, argumentName,
    ↳ messageBuilder);
234
235 /// <summary>
236 /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed only for DEBUG build configuration.</para>
237 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется только для конфигурации сборки DEBUG.</para>
238 /// </summary>
239 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
240 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
241 /// <param name="argument"></param>
242 /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↳ диапазона.</para></param>
243 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
244 /// <param name="message"><para>The message of the thrown
    ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
245 [Conditional("DEBUG")]
246 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argument, Range<TArgument> range, string argumentName, string message) =>
    ↳ Ensure.Always.ArgumentInRange(argument, range, argumentName, message);
247
248 /// <summary>
249 /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed only for DEBUG build configuration.</para>
250 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется только для конфигурации сборки DEBUG.</para>
251 /// </summary>
252 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
253 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
254 /// <param name="argument"></param>
255 /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↳ диапазона.</para></param>
256 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
257 [Conditional("DEBUG")]
258 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argument, Range<TArgument> range, string argumentName) =>
    ↳ Ensure.Always.ArgumentInRange(argument, range, argumentName);
259
260 /// <summary>

```

```

261 /// <para>Ensures that the argument value is in the specified range. This check is
262   → performed only for DEBUG build configuration.</para>
263 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
264   → проверка выполняется только для конфигурации сборки DEBUG.</para>
265 /// </summary>
266 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
267   → аргумента.</para></typeparam>
268 /// <param name="root"><para>The extension root to which this method is
269   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
270 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
271   → аргумента.</para></param>
272 /// <param name="minimum"><para>The minimum possible argument's
273   → value.</para><para>Минимально возможное значение аргумента.</para></param>
274 /// <param name="maximum"><para>The maximum possible argument's
275   → value.</para><para>Максимально возможное значение аргумента.</para></param>
276 [Conditional("DEBUG")]
277 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
278   → TArgument argumentValue, TArgument minimum, TArgument maximum) =>
279   → Ensure.Always.ArgumentInRange(argumentValue, new Range<TArgument>(minimum, maximum),
280   → null);
281
282 /// <summary>
283 /// <para>Ensures that the argument value is in the specified range. This check is
284   → performed only for DEBUG build configuration.</para>
285 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
286   → проверка выполняется только для конфигурации сборки DEBUG.</para>
287 /// </summary>
288 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
289   → аргумента.</para></typeparam>
290 /// <param name="root"><para>The extension root to which this method is
291   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
292 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
293   → аргумента.</para></param>
294 /// <param name="minimum"><para>The minimum possible argument's
295   → value.</para><para>Минимально возможное значение аргумента.</para></param>
296 /// <param name="maximum"><para>The maximum possible argument's
297   → value.</para><para>Максимально возможное значение аргумента.</para></param>
298 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
299   → аргумента.</para></param>
300 [Conditional("DEBUG")]
301 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
302   → TArgument argumentValue, TArgument minimum, TArgument maximum, string argumentName)
303   → => Ensure.Always.ArgumentInRange(argumentValue, new Range<TArgument>(minimum,
304   → maximum), argumentName);
305
306 /// <summary>
307 /// <para>Ensures that the argument value is in the specified range. This check is
308   → performed only for DEBUG build configuration.</para>
309 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
310   → проверка выполняется только для конфигурации сборки DEBUG.</para>
311 /// </summary>
312 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
313   → аргумента.</para></typeparam>
314 /// <param name="root"><para>The extension root to which this method is
315   → bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
316 /// <param name="argument"></param>
317 /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
318   → диапазона.</para></param>
319 [Conditional("DEBUG")]
320 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
321   → TArgument argument, Range<TArgument> range) =>
322   → Ensure.Always.ArgumentInRange(argument, range, null);
323
324 #endregion
325 }
326 }

```

./Range.cs

```

1 using System;
2 using System.Collections.Generic;
3 using Platform.Exceptions;
4
5 namespace Platform.Ranges
6 {
7     /// <summary>
8     /// <para>Represents a range between minumum and maximum values.</para>

```

```

9  /// <para>Представляет диапазон между минимальным и максимальным значениями.</para>
10 /// </summary>
11 /// <remarks>
12 /// <para>Based on <a href="http://stackoverflow.com/questions/5343006/is-there-a-c-sharp-type-for-representing-an-integer-range">the question at
   ↳ pe-for-representing-an-integer-range</a> StackOveflow</a>.</para>
13 /// <para>Основано на <a href="http://stackoverflow.com/questions/5343006/is-there-a-c-sharp-type-for-representing-an-integer-range">вопросе в
   ↳ StackOveflow</a>.</para>
14 /// </remarks>
15 public struct Range<T> : IEquatable<Range<T>>
16 {
17     private static readonly Comparer<T> _comparer = Comparer<T>.Default;
18     private static readonly EqualityComparer<T> _equalityComparer =
19         ↳ EqualityComparer<T>.Default;
20
21     /// <summary>
22     /// <para>Returns minimum value of the range.</para>
23     /// <para>Возвращает минимальное значение диапазона.</para>
24     /// </summary>
25     public readonly T Minimum;
26
27     /// <summary>
28     /// <para>Returns maximum value of the range.</para>
29     /// <para>Возвращает максимальное значение диапазона.</para>
30     /// </summary>
31     public readonly T Maximum;
32
33     /// <summary>
34     /// <para>Initializes a new instance of the Range class.</para>
35     /// <para>Инициализирует новый экземпляр класса Range.</para>
36     /// </summary>
37     /// <param name="minimumAndMaximum"><para>Single value for both Minimum and Maximum
   ↳ fields.</para><para>Одно значение для полей Minimum и Maximum.</para></param>
38     public Range(T minimumAndMaximum)
39     {
40         Minimum = minimumAndMaximum;
41         Maximum = minimumAndMaximum;
42     }
43
44     /// <summary>
45     /// <para>Initializes a new instance of the Range class.</para>
46     /// <para>Инициализирует новый экземпляр класса Range.</para>
47     /// </summary>
48     /// <param name="minimum"><para>The minimum value of the range.</para><para>Минимальное
   ↳ значение диапазона.</para></param>
49     /// <param name="maximum"><para>The maximum value of the range.</para><para>Максимальное
   ↳ значение диапазона.</para></param>
50     /// <exception cref="ArgumentException"><para>Thrown when the maximum is less than the
   ↳ minimum.</para><para>Выбрасывается, когда максимум меньше
   ↳ минимума.</para></exception>
51     public Range(T minimum, T maximum)
52     {
53         Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
54             ↳ nameof(maximum));
55         Minimum = minimum;
56         Maximum = maximum;
57     }
58
59     /// <summary>
60     /// <para>Presents the Range in readable format.</para>
61     /// <para>Представляет диапазон в удобном для чтения формате.</para>
62     /// </summary>
63     /// <returns><para>String representation of the Range.</para><para>Строковое
   ↳ представление диапазона.</para></returns>
64     public override string ToString() => $"{Minimum}, {Maximum}";
65
66     /// <summary>
67     /// <para>Determines if the provided value is inside the range.</para>
68     /// <para>Определяет, находится ли указанное значение внутри диапазона.</para>
69     /// </summary>
70     /// <param name="value"><para>The value to test.</para><para>Значение для
   ↳ проверки.</para></param>
71     /// <returns><para>True if the value is inside Range, else false.</para><para>True, если
   ↳ значение находится внутри диапазона, иначе false.</para></returns>
72     public bool ContainsValue(T value) => _comparer.Compare(Minimum, value) <= 0 &&
73         ↳ _comparer.Compare(Maximum, value) >= 0;

```



```

72     /// <summary>
73     /// <para>Determines if this Range is inside the bounds of another range.</para>
74     /// <para>Определяет, находится ли этот диапазон в пределах другого диапазона.</para>
75     /// </summary>
76     /// <param name="range"><para>The parent range to test on.</para><para>Родительский
    ↪ диапазон для проверки.</para></param>
77     /// <returns><para>True if range is inclusive, else false.</para><para>True, если
    ↪ диапазон включен, иначе false.</para></returns>
    public bool IsInsideRange(Range<T> range) => range.ContainsRange(this);
79
80     /// <summary>
81     /// <para>Determines if another range is inside the bounds of this range.</para>
82     /// <para>Определяет, находится ли другой диапазон внутри границ этого диапазона.</para>
83     /// </summary>
84     /// <param name="range"><para>The child range to test.</para><para>Дочерний диапазон для
    ↪ проверки.</para></param>
85     /// <returns><para>True if range is inside, else false.</para><para>True, если диапазон
    ↪ находится внутри, иначе false.</para></returns>
    public bool ContainsRange(Range<T> range) => ContainsValue(range.Minimum) &&
    ↪ ContainsValue(range.Maximum);
87
88     /// <summary>
89     /// <para>Indicates whether the current range is equal to another range.</para>
90     /// <para>Определяет, равен ли текущий диапазон другому диапазону.</para>
91     /// </summary>
92     /// <param name="other"><para>A range to compare with this range.</para><para>Диапазон
    ↪ для сравнения с этим диапазоном.</para></param>
93     /// <returns><para>True if the current range is equal to the other range; otherwise,
    ↪ false.</para><para>True, если текущий диапазон равен другому диапазону; иначе
    ↪ false.</para></returns>
    public bool Equals(Range<T> other) => _equalityComparer.Equals(Minimum, other.Minimum)
    ↪ && _equalityComparer.Equals(Maximum, other.Maximum);
95
96     /// <summary>
97     /// <para>Creates a new <see cref="ValueTuple{T,T}"> struct initialized with <see
    ↪ cref="Range{T}.Minimum"/> as <see cref="ValueTuple{T,T}.Item1"/> and <see
    ↪ cref="Range{T}.Maximum"/> as <see cref="ValueTuple{T,T}.Item2"/>.</para>
98     /// <para>Создает новую структуру <see cref="ValueTuple{T,T}">, инициализированную с
    ↪ помощью <see cref="Range{T}.Minimum"/> как <see cref="ValueTuple{T,T}.Item1"/> и
    ↪ <see cref="Range{T}.Maximum"/> как <see cref="ValueTuple{T,T}.Item2"/>.</para>
99     /// </summary>
100    /// <param name="range"><para>The range of <typeparamref
    ↪ name="T"/>.</para><para>Диапазон значений <typeparamref name="T"/>.</para></param>
101    public static implicit operator ValueTuple<T, T>(Range<T> range) => (range.Minimum,
    ↪ range.Maximum);
102
103    /// <summary>
104    /// <para>Creates a new <see cref="Range{T}"> struct initialized with <see
    ↪ cref="ValueTuple{T,T}.Item1"/> as <see cref="Range{T}.Minimum"/> and <see
    ↪ cref="ValueTuple{T,T}.Item2"/> as <see cref="Range{T}.Maximum"/>.</para>
105    /// <para>Создает новую структуру <see cref="Range{T}">, инициализированную с помощью
    ↪ <see cref="ValueTuple{T,T}.Item1"/> как <see cref="Range{T}.Minimum"/> и <see
    ↪ cref="ValueTuple{T,T}.Item2"/> как <see cref="Range{T}.Maximum"/>.</para>
106    /// </summary>
107    /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
108    public static implicit operator Range<T>(ValueTuple<T, T> tuple) => new
    ↪ Range<T>(tuple.Item1, tuple.Item2);
109 }
110 }

```

./RangeExtensions.cs

```

1 namespace Platform.Ranges
2 {
3     /// <summary>
4     /// <para>Provides a set of extension methods for <see cref="Range{T}"> structs.</para>
5     /// <para>Предоставляет набор методов расширения для структур <see cref="Range{T}">.</para>
6     /// </summary>
7     public static class RangeExtensions
8     {
9         /// <summary>
10        /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↪ cref="Range{T}.Maximum"/>.</para>
11        /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↪ cref="Range{T}.Maximum"/>.</para>
12        /// </summary>

```

```

13  /// <param name="range"><para>The range of <see cref="ulong"/>.</para><para>Диапазон
    ↳ значений <see cref="ulong"/>.</para></param>
14  /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
    ↳ и <see cref="Range{T}.Maximum"/>.</para></returns>
15  public static ulong Difference(this Range<ulong> range) => range.Maximum - range.Minimum;
16
17  /// <summary>
18  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
19  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
20  /// </summary>
21  /// <param name="range"><para>The range of <see cref="uint"/>.</para><para>Диапазон
    ↳ значений <see cref="uint"/>.</para></param>
22  /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
    ↳ и <see cref="Range{T}.Maximum"/>.</para></returns>
23  public static uint Difference(this Range<uint> range) => range.Maximum - range.Minimum;
24
25  /// <summary>
26  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
27  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
28  /// </summary>
29  /// <param name="range"><para>The range of <see cref="ushort"/>.</para><para>Диапазон
    ↳ значений <see cref="ushort"/>.</para></param>
30  /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
    ↳ и <see cref="Range{T}.Maximum"/>.</para></returns>
31  public static ushort Difference(this Range<ushort> range) => (ushort)(range.Maximum -
    ↳ range.Minimum);
32
33  /// <summary>
34  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
35  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
36  /// </summary>
37  /// <param name="range"><para>The range of <see cref="byte"/>.</para><para>Диапазон
    ↳ значений <see cref="byte"/>.</para></param>
38  /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
    ↳ и <see cref="Range{T}.Maximum"/>.</para></returns>
39  public static byte Difference(this Range<byte> range) => (byte)(range.Maximum -
    ↳ range.Minimum);
40
41  /// <summary>
42  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
43  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
44  /// </summary>
45  /// <param name="range"><para>The range of <see cref="long"/>.</para><para>Диапазон
    ↳ значений <see cref="long"/>.</para></param>
46  /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
    ↳ и <see cref="Range{T}.Maximum"/>.</para></returns>
47  public static long Difference(this Range<long> range) => range.Maximum - range.Minimum;
48
49  /// <summary>
50  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
51  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
52  /// </summary>
53  /// <param name="range"><para>The range of <see cref="int"/>.</para><para>Диапазон
    ↳ значений <see cref="int"/>.</para></param>
54  /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
    ↳ и <see cref="Range{T}.Maximum"/>.</para></returns>
55  public static int Difference(this Range<int> range) => range.Maximum - range.Minimum;
56
57  /// <summary>

```

```

58 /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
59   → cref="Range{T}.Maximum"/>.</para>
60 /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
61   → cref="Range{T}.Maximum"/>.</para>
62 /// </summary>
63 /// <param name="range"><para>The range of <see cref="short"/>.</para><para>Диапазон
64   → значений <see cref="short"/>.</para></param>
65 /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
66   → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
67   → и <see cref="Range{T}.Maximum"/>.</para></returns>
68 public static short Difference(this Range<short> range) => (short)(range.Maximum -
69   → range.Minimum);
70
71 /// <summary>
72 /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
73   → cref="Range{T}.Maximum"/>.</para>
74 /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
75   → cref="Range{T}.Maximum"/>.</para>
76 /// </summary>
77 /// <param name="range"><para>The range of <see cref="sbyte"/>.</para><para>Диапазон
78   → значений <see cref="sbyte"/>.</para></param>
79 /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
80   → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
81   → и <see cref="Range{T}.Maximum"/>.</para></returns>
82 public static sbyte Difference(this Range<sbyte> range) => (sbyte)(range.Maximum -
83   → range.Minimum);
84
85 /// <summary>
86 /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
87   → cref="Range{T}.Maximum"/>.</para>
88 /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
89   → cref="Range{T}.Maximum"/>.</para>
90 /// </summary>
91 /// <param name="range"><para>The range of <see cref="double"/>.</para><para>Диапазон
92   → значений <see cref="double"/>.</para></param>
93 /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
94   → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
95   → и <see cref="Range{T}.Maximum"/>.</para></returns>
96 public static double Difference(this Range<double> range) => range.Maximum -
97   → range.Minimum;
98
99 /// <summary>
100 /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
101   → cref="Range{T}.Maximum"/>.</para>
102 /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
103   → cref="Range{T}.Maximum"/>.</para>
104 /// </summary>
105 /// <param name="range"><para>The range of <see cref="float"/>.</para><para>Диапазон
106   → значений <see cref="float"/>.</para></param>
107 /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
108   → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
109   → и <see cref="Range{T}.Maximum"/>.</para></returns>
110 public static float Difference(this Range<float> range) => range.Maximum - range.Minimum;
111
112 /// <summary>
113 /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
114   → cref="Range{T}.Maximum"/>.</para>
115 /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
116   → cref="Range{T}.Maximum"/>.</para>
117 /// </summary>
118 /// <param name="range"><para>The range of <see cref="decimal"/>.</para><para>Диапазон
119   → значений <see cref="decimal"/>.</para></param>
120 /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
121   → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
122   → и <see cref="Range{T}.Maximum"/>.</para></returns>
123 public static decimal Difference(this Range<decimal> range) => range.Maximum -
124   → range.Minimum;
125
126 }
127 }

```

Index

./EnsureExtensions.cs, 1

./Range.cs, 7

./RangeExtensions.cs, 9