

# LinksPlatform's Platform.Ranges Class Library

## 1.1 ./csharp/Platform.Ranges/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.Contains(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>
210  /// <param name="maximumArgument"><para>The argument with the maximum
211  → value.</para><para>Аргумент с максимальным значением.</para></param>
212  /// <param name="maximumArgumentName"><para>The name of argument with the maximum
213  → 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 }

```

## 1.2 ./csharp/Platform.Ranges/Range.cs

```

1 namespace Platform.Ranges
2 {
3     /// <summary>
4     /// <para>Contains static fields with <see cref="Range{T}"/> constants.</para>
5     /// <para>Содержит статические поля с константами типа <see cref="Range{T}"/>.</para>
6     /// </summary>
7     public static class Range
8     {

```

```

9      /// <summary>
10     /// <para>Gets the whole <see cref="sbyte"/> values range.</para>
11     /// <para>Возвращает весь диапазон значений <see cref="sbyte"/>.</para>
12     /// </summary>
13     public static readonly Range<sbyte> SByte = new Range<sbyte>(sbyte.MinValue,
14         ↪ sbyte.MaxValue);
15
16     /// <summary>
17     /// <para>Gets the whole <see cref="short"/> values range.</para>
18     /// <para>Возвращает весь диапазон значений <see cref="short"/>.</para>
19     /// </summary>
20     public static readonly Range<short> Int16 = new Range<short>(short.MinValue,
21         ↪ short.MaxValue);
22
23     /// <summary>
24     /// <para>Gets the whole <see cref="int"/> values range.</para>
25     /// <para>Возвращает весь диапазон значений <see cref="int"/>.</para>
26     /// </summary>
27     public static readonly Range<int> Int32 = new Range<int>(int.MinValue, int.MaxValue);
28
29     /// <summary>
30     /// <para>Gets the whole <see cref="long"/> values range.</para>
31     /// <para>Возвращает весь диапазон значений <see cref="long"/>.</para>
32     /// </summary>
33     public static readonly Range<long> Int64 = new Range<long>(long.MinValue, long.MaxValue);
34
35     /// <summary>
36     /// <para>Gets the whole <see cref="byte"/> values range.</para>
37     /// <para>Возвращает весь диапазон значений <see cref="byte"/>.</para>
38     /// </summary>
39     public static readonly Range<byte> Byte = new Range<byte>(byte.MinValue, byte.MaxValue);
40
41     /// <summary>
42     /// <para>Gets the whole <see cref="ushort"/> values range.</para>
43     /// <para>Возвращает весь диапазон значений <see cref="ushort"/>.</para>
44     /// </summary>
45     public static readonly Range<ushort> UInt16 = new Range<ushort>(ushort.MinValue,
46         ↪ ushort.MaxValue);
47
48     /// <summary>
49     /// <para>Gets the whole <see cref="uint"/> values range.</para>
50     /// <para>Возвращает весь диапазон значений <see cref="uint"/>.</para>
51     /// </summary>
52     public static readonly Range<uint> UInt32 = new Range<uint>(uint.MinValue,
53         ↪ uint.MaxValue);
54
55     /// <summary>
56     /// <para>Gets the whole <see cref="ulong"/> values range.</para>
57     /// <para>Возвращает весь диапазон значений <see cref="ulong"/>.</para>
58     /// </summary>
59     public static readonly Range<ulong> UInt64 = new Range<ulong>(ulong.MinValue,
60         ↪ ulong.MaxValue);
61
62     /// <summary>
63     /// <para>Gets the whole <see cref="float"/> values range.</para>
64     /// <para>Возвращает весь диапазон значений <see cref="float"/>.</para>
65     /// </summary>
66     public static readonly Range<float> Single = new Range<float>(float.MinValue,
67         ↪ float.MaxValue);
68
69     /// <summary>
70     /// <para>Gets the whole <see cref="double"/> values range.</para>
71     /// <para>Возвращает весь диапазон значений <see cref="double"/>.</para>
72     /// </summary>
73     public static readonly Range<double> Double = new Range<double>(double.MinValue,
74         ↪ double.MaxValue);
75
76     /// <summary>
77     /// <para>Gets the whole <see cref="decimal"/> values range.</para>
78     /// <para>Возвращает весь диапазон значений <see cref="decimal"/>.</para>
79     /// </summary>
80     public static readonly Range<decimal> Decimal = new Range<decimal>(decimal.MinValue,
81         ↪ decimal.MaxValue);
82
83     }
84 }

```



### 1.3 ./csharp/Platform.Ranges/RangeExtensions.cs

```
1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Ranges
4 {
5     /// <summary>
6     /// <para>Provides a set of extension methods for <see cref="Range{T}"/> structs.</para>
7     /// <para>Предоставляет набор методов расширения для структур <see cref="Range{T}"/>.</para>
8     /// </summary>
9     public static class RangeExtensions
10     {
11         /// <summary>
12         /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
13         → cref="Range{T}.Maximum"/>.</para>
14         /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
15         → cref="Range{T}.Maximum"/>.</para>
16         /// </summary>
17         /// <param name="range"><para>The range of <see cref="ulong"/>.</para><para>Диапазон
18         → значений <see cref="ulong"/>.</para></param>
19         /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
20         → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
21         → и <see cref="Range{T}.Maximum"/>.</para></returns>
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public static ulong Difference(this Range<ulong> range) => range.Maximum - range.Minimum;
24
25         /// <summary>
26         /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
27         → cref="Range{T}.Maximum"/>.</para>
28         /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
29         → cref="Range{T}.Maximum"/>.</para>
30         /// </summary>
31         /// <param name="range"><para>The range of <see cref="uint"/>.</para><para>Диапазон
32         → значений <see cref="uint"/>.</para></param>
33         /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
34         → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
35         → и <see cref="Range{T}.Maximum"/>.</para></returns>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static uint Difference(this Range<uint> range) => range.Maximum - range.Minimum;
38
39         /// <summary>
40         /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
41         → cref="Range{T}.Maximum"/>.</para>
42         /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
43         → cref="Range{T}.Maximum"/>.</para>
44         /// </summary>
45         /// <param name="range"><para>The range of <see cref="ushort"/>.</para><para>Диапазон
46         → значений <see cref="ushort"/>.</para></param>
47         /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
48         → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
49         → и <see cref="Range{T}.Maximum"/>.</para></returns>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public static ushort Difference(this Range<ushort> range) => (ushort)(range.Maximum -
52         → range.Minimum);
53
54         /// <summary>
55         /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
56         → cref="Range{T}.Maximum"/>.</para>
57         /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
58         → cref="Range{T}.Maximum"/>.</para>
59         /// </summary>
60         /// <param name="range"><para>The range of <see cref="byte"/>.</para><para>Диапазон
61         → значений <see cref="byte"/>.</para></param>
62         /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
63         → cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
64         → и <see cref="Range{T}.Maximum"/>.</para></returns>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public static byte Difference(this Range<byte> range) => (byte)(range.Maximum -
67         → range.Minimum);
68
69         /// <summary>
70         /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
71         → cref="Range{T}.Maximum"/>.</para>
72         /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
73         → cref="Range{T}.Maximum"/>.</para>
74         /// </summary>
```

```

51  /// <param name="range"><para>The range of <see cref="long"/>.</para><para>Диапазон
    ↳ значений <see cref="long"/>.</para></param>
52  /// <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>
53  [MethodImpl(MethodImplOptions.AggressiveInlining)]
54  public static long Difference(this Range<long> range) => range.Maximum - range.Minimum;
55
56  /// <summary>
57  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
58  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
59  /// </summary>
60  /// <param name="range"><para>The range of <see cref="int"/>.</para><para>Диапазон
    ↳ значений <see cref="int"/>.</para></param>
61  /// <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>
62  [MethodImpl(MethodImplOptions.AggressiveInlining)]
63  public static int Difference(this Range<int> range) => range.Maximum - range.Minimum;
64
65  /// <summary>
66  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
67  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
68  /// </summary>
69  /// <param name="range"><para>The range of <see cref="short"/>.</para><para>Диапазон
    ↳ значений <see cref="short"/>.</para></param>
70  /// <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>
71  [MethodImpl(MethodImplOptions.AggressiveInlining)]
72  public static short Difference(this Range<short> range) => (short)(range.Maximum -
    ↳ range.Minimum);
73
74  /// <summary>
75  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
76  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
77  /// </summary>
78  /// <param name="range"><para>The range of <see cref="sbyte"/>.</para><para>Диапазон
    ↳ значений <see cref="sbyte"/>.</para></param>
79  /// <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>
80  [MethodImpl(MethodImplOptions.AggressiveInlining)]
81  public static sbyte Difference(this Range<sbyte> range) => (sbyte)(range.Maximum -
    ↳ range.Minimum);
82
83  /// <summary>
84  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
85  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
86  /// </summary>
87  /// <param name="range"><para>The range of <see cref="double"/>.</para><para>Диапазон
    ↳ значений <see cref="double"/>.</para></param>
88  /// <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>
89  [MethodImpl(MethodImplOptions.AggressiveInlining)]
90  public static double Difference(this Range<double> range) => range.Maximum -
    ↳ range.Minimum;
91
92  /// <summary>
93  /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↳ cref="Range{T}.Maximum"/>.</para>
94  /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↳ cref="Range{T}.Maximum"/>.</para>
95  /// </summary>
96  /// <param name="range"><para>The range of <see cref="float"/>.</para><para>Диапазон
    ↳ значений <see cref="float"/>.</para></param>

```

```

97     /// <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>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public static float Difference(this Range<float> range) => range.Maximum - range.Minimum;
100
101     /// <summary>
102     /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
    ↪   cref="Range{T}.Maximum"/>.</para>
103     /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
    ↪   cref="Range{T}.Maximum"/>.</para>
104     /// </summary>
105     /// <param name="range"><para>The range of <see cref="decimal"/>.</para><para>Диапазон
    ↪   значений <see cref="decimal"/>.</para></param>
106     /// <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>
107     [MethodImpl(MethodImplOptions.AggressiveInlining)]
108     public static decimal Difference(this Range<decimal> range) => range.Maximum -
    ↪   range.Minimum;
109 }
110 }

```

#### 1.4 ./csharp/Platform.Ranges/Range[T].cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Exceptions;
5
6  namespace Platform.Ranges
7  {
8      /// <summary>
9      /// <para>Represents a range between minumum and maximum values.</para>
10     /// <para>Представляет диапазон между минимальным и максимальным значениями.</para>
11     /// </summary>
12     /// <remarks>
13     /// <para>Based on <a href="http://stackoverflow.com/questions/5343006/is-there-a-c-sharp-ty_
    ↪   pe-for-representing-an-integer-range">the question at
    ↪   StackOveflow</a>.</para>
14     /// <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>
15     /// </remarks>
16     public struct Range<T> : IEquatable<Range<T>>
17     {
18         private static readonly Comparer<T> _comparer = Comparer<T>.Default;
19         private static readonly EqualityComparer<T> _equalityComparer =
    ↪   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         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public Range(T minimumAndMaximum)
40         {
41             Minimum = minimumAndMaximum;
42             Maximum = minimumAndMaximum;
43         }
44
45         /// <summary>
46         /// <para>Initializes a new instance of the Range class.</para>
47         /// <para>Инициализирует новый экземпляр класса Range.</para>
48         /// </summary>

```

```

49  /// <param name="minimum"><para>The minimum value of the range.</para><para>Минимальное
    ↳ значение диапазона.</para></param>
50  /// <param name="maximum"><para>The maximum value of the range.</para><para>Максимальное
    ↳ значение диапазона.</para></param>
51  /// <exception cref="ArgumentException"><para>Thrown when the maximum is less than the
    ↳ minimum.</para><para>Выбрасывается, когда максимум меньше
    ↳ минимума.</para></exception>
52  [MethodImpl(MethodImplOptions.AggressiveInlining)]
53  public Range(T minimum, T maximum)
54  {
55      Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
        ↳ nameof(maximum));
56      Minimum = minimum;
57      Maximum = maximum;
58  }
59
60  /// <summary>
61  /// <para>Presents the Range in readable format.</para>
62  /// <para>Представляет диапазон в удобном для чтения формате.</para>
63  /// </summary>
64  /// <returns><para>String representation of the Range.</para><para>Строковое
    ↳ представление диапазона.</para></returns>
65  [MethodImpl(MethodImplOptions.AggressiveInlining)]
66  public override string ToString() => $"{Minimum}, {Maximum}";
67
68  /// <summary>
69  /// <para>Determines if the provided value is inside the range.</para>
70  /// <para>Определяет, находится ли указанное значение внутри диапазона.</para>
71  /// </summary>
72  /// <param name="value"><para>The value to test.</para><para>Значение для
    ↳ проверки.</para></param>
73  /// <returns><para>True if the value is inside Range, else false.</para><para>True, если
    ↳ значение находится внутри диапазона, иначе false.</para></returns>
74  [MethodImpl(MethodImplOptions.AggressiveInlining)]
75  public bool Contains(T value) => _comparer.Compare(Minimum, value) <= 0 &&
    ↳ _comparer.Compare(Maximum, value) >= 0;
76
77  /// <summary>
78  /// <para>Determines if another range is inside the bounds of this range.</para>
79  /// <para>Определяет, находится ли другой диапазон внутри границ этого диапазона.</para>
80  /// </summary>
81  /// <param name="range"><para>The child range to test.</para><para>Дочерний диапазон для
    ↳ проверки.</para></param>
82  /// <returns><para>True if range is inside, else false.</para><para>True, если диапазон
    ↳ находится внутри, иначе false.</para></returns>
83  [MethodImpl(MethodImplOptions.AggressiveInlining)]
84  public bool Contains(Range<T> range) => Contains(range.Minimum) &&
    ↳ Contains(range.Maximum);
85
86  /// <summary>
87  /// <para>Determines whether the current range is equal to another range.</para>
88  /// <para>Определяет, равен ли текущий диапазон другому диапазону.</para>
89  /// </summary>
90  /// <param name="other"><para>A range to compare with this range.</para><para>Диапазон
    ↳ для сравнения с этим диапазоном.</para></param>
91  /// <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>
92  [MethodImpl(MethodImplOptions.AggressiveInlining)]
93  public bool Equals(Range<T> other) => _equalityComparer.Equals(Minimum, other.Minimum)
    ↳ && _equalityComparer.Equals(Maximum, other.Maximum);
94
95  /// <summary>
96  /// <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>
97  /// <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>
98  /// </summary>
99  /// <param name="range"><para>The range of <typeparamref
    ↳ name="T"/>.</para><para>Диапазон значений <typeparamref name="T"/>.</para></param>
100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
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 [MethodImpl(MethodImplOptions.AggressiveInlining)]
109 public static implicit operator Range<T>(ValueTuple<T, T> tuple) => new
    ↳ Range<T>(tuple.Item1, tuple.Item2);
110
111 /// <summary>
112 /// <para>Determines whether the current range is equal to another object.</para>
113 /// <para>Определяет, равен ли текущий диапазон другому объекту.</para>
114 /// </summary>
115 /// <param name="obj"><para>An object to compare with this range.</para><para>Объект для
    ↳ сравнения с этим диапазоном.</para></param>
116 /// <returns><para>True if the current range is equal to the other object; otherwise,
    ↳ false.</para><para>True, если текущий диапазон равен другому объекту; иначе
    ↳ false.</para></returns>
117 [MethodImpl(MethodImplOptions.AggressiveInlining)]
118 public override bool Equals(object obj) => obj is Range<T> range ? Equals(range) : false;
119
120 /// <summary>
121 /// Calculates the hash code for the current <see cref="Range{T}"> instance.
122 /// </summary>
123 /// <returns>The hash code for the current <see cref="Range{T}"> instance.</returns>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 public override int GetHashCode() => (Minimum, Maximum).GetHashCode();
126
127 /// <summary>
128 /// <para>Determines if the specified range is equal to the current range.</para>
129 /// <para>Определяет, равен ли указанный диапазон текущему диапазону.</para>
130 /// </summary>
131 /// <param name="left"><para>The current range.</para><para>Текущий
    ↳ диапазон.</para></param>
132 /// <param name="right"><para>A range to compare with this range.</para><para>Диапазон
    ↳ для сравнения с этим диапазоном.</para></param>
133 /// <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>
134 [MethodImpl(MethodImplOptions.AggressiveInlining)]
135 public static bool operator ==(Range<T> left, Range<T> right) => left.Equals(right);
136
137 /// <summary>
138 /// <para>Determines if the specified range is not equal to the current range.</para>
139 /// <para>Определяет, не равен ли указанный диапазон текущему диапазону.</para>
140 /// </summary>
141 /// <param name="left"><para>The current range.</para><para>Текущий
    ↳ диапазон.</para></param>
142 /// <param name="right"><para>A range to compare with this range.</para><para>Диапазон
    ↳ для сравнения с этим диапазоном.</para></param>
143 /// <returns><para>True if the current range is not equal to the other range; otherwise,
    ↳ false.</para><para>True, если текущий диапазон не равен другому диапазону; иначе
    ↳ false.</para></returns>
144 [MethodImpl(MethodImplOptions.AggressiveInlining)]
145 public static bool operator !=(Range<T> left, Range<T> right) => !(left == right);
146 }
147 }

```

## 1.5 ./csharp/Platform.Ranges.Tests/EnsureExtensionsTests.cs

```

1 using System;
2 using Xunit;
3 using Platform.Exceptions;
4
5 namespace Platform.Ranges.Tests
6 {
7     public static class EnsureExtensionsTests
8     {
9         [Fact]
10         public static void MaximumArgumentIsGreaterOrEqualToMinimumExceptionTest() =>
            ↳ Assert.Throws<ArgumentException>(() =>
            ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(2, 1));
11
12         [Fact]

```

```

13     public static void ArgumentInRangeExceptionTest() =>
14         ↪ Assert.Throws<ArgumentOutOfRangeException>(() => Ensure.Always.ArgumentInRange(5,
15         ↪ (6, 7)));
16     }
17 }

```

## 1.6 ./csharp/Platform.Ranges.Tests/RangeTests.cs

```

1  using System;
2  using Xunit;
3
4  namespace Platform.Ranges.Tests
5  {
6      public static class RangeTests
7      {
8          [Fact]
9          public static void ConstructorsTest()
10         {
11             var range1 = new Range<int>(1, 3);
12             Assert.Equal(1, range1.Minimum);
13             Assert.Equal(3, range1.Maximum);
14             Assert.Throws<ArgumentException>(() => new Range<int>(2, 1));
15             var range2 = new Range<int>(5);
16             Assert.Equal(5, range2.Minimum);
17             Assert.Equal(5, range2.Maximum);
18         }
19
20         [Fact]
21         public static void ContainsTest()
22         {
23             var range = new Range<int>(1, 3);
24             Assert.True(range.Contains(1));
25             Assert.True(range.Contains(2));
26             Assert.True(range.Contains(3));
27             Assert.True(range.Contains((2, 3)));
28             Assert.False(range.Contains((3, 4)));
29         }
30
31         [Fact]
32         public static void DifferenceTest()
33         {
34             var range = new Range<int>(1, 3);
35             Assert.Equal(2, range.Difference());
36         }
37
38         [Fact]
39         public static void ToStringTest()
40         {
41             var range = new Range<int>(1, 3);
42             Assert.Equal("[1, 3]", range.ToString());
43         }
44
45         [Fact]
46         public static void EqualityTest()
47         {
48             var range1 = new Range<int>(1, 3);
49             var range1Duplicate = new Range<int>(1, 3);
50             var range2 = new Range<int>(2, 5);
51             Assert.True(range1 == range1Duplicate);
52             Assert.Equal(range1, range1Duplicate);
53             Assert.True(range1 != range2);
54             Assert.NotEqual(range1, range2);
55         }
56     }
57 }

```

## Index

- ./csharp/Platform.Ranges.Tests/EnsureExtensionsTests.cs, 13
- ./csharp/Platform.Ranges.Tests/RangeTests.cs, 14
- ./csharp/Platform.Ranges/EnsureExtensions.cs, 1
- ./csharp/Platform.Ranges/Range.cs, 7
- ./csharp/Platform.Ranges/RangeExtensions.cs, 8
- ./csharp/Platform.Ranges/Range[T].cs, 11