

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         /// <summary>
21         /// <para>
22         ///   The default maximum should be greater or equal to minimum message.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         private const string DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage = "Maximum
27         ↳ should be greater or equal to minimum.";
28
29         #region Always
30
31         /// <summary>
32         /// <para>Ensures that the argument with the maximum value is greater than or equal to
33         ///   ↳ the minimum value. This check is performed regardless of the build
34         ///   ↳ configuration.</para>
35         /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
36         ///   ↳ минимальному значению. Эта проверка выполняется независимо от конфигурации
37         ///   ↳ сборки.</para>
38         /// </summary>
39         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
40         ///   ↳ аргумента.</para></typeparam>
41         /// <param name="root"><para>The extension root to which this method is
42         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
43         /// <param name="minimumArgument"><para>The argument with the minimum
44         ///   ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
45         /// <param name="maximumArgument"><para>The argument with the maximum
46         ///   ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
47         /// <param name="maximumArgumentName"><para>The name of argument with the maximum
48         ///   ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>
49         /// <param name="messageBuilder"><para>The thrown exception's message building <see
50         ///   ↳ cref="Func{String}"/>.</para><para>Собирающая сообщение для выбрасываемого
51         ///   ↳ исключения <see cref="Func{String}"/>.</para></param>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
54         ↳ EnsureAlwaysExtensionRoot root, TArgument minimumArgument, TArgument
55         ↳ maximumArgument, string maximumArgumentName, Func<string> messageBuilder)
56         {
57             if (Comparer<TArgument>.Default.Compare(maximumArgument, minimumArgument) < 0)
58             {
59                 throw new ArgumentException(messageBuilder(), maximumArgumentName);
60             }
61         }
62
63         /// <summary>
64         /// <para>Ensures that the argument with the maximum value is greater than or equal to
65         ///   ↳ the minimum value. This check is performed regardless of the build
66         ///   ↳ configuration.</para>
67         /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
68         ///   ↳ минимальному значению. Эта проверка выполняется независимо от конфигурации
69         ///   ↳ сборки.</para>
70         /// </summary>
71         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
72         ///   ↳ аргумента.</para></typeparam>
73         /// <param name="root"><para>The extension root to which this method is
74         ///   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
```

```

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

```

```

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

```

```

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

```

```

185 /// <param name="minimumArgument"><para>The argument with the minimum
186   ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
187 /// <param name="maximumArgument"><para>The argument with the maximum
188   ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
189 /// <param name="maximumArgumentName"><para>The name of argument with the maximum
190   ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>
191 /// <param name="messageBuilder"><para>The thrown exception's message building <see
192   ↳ cref="Func{String}"/>.</para><para>Собирающая сообщения для выбрасываемого
193   ↳ исключения <see cref="Func{String}"/>.</para></param>
194 [Conditional("DEBUG")]
195 public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
196   ↳ EnsureOnDebugExtensionRoot root, TArgument minimumArgument, TArgument
197   ↳ maximumArgument, string maximumArgumentName, Func<string> messageBuilder) =>
198   ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimumArgument,
199   ↳ maximumArgument, maximumArgumentName, messageBuilder);
200
201 /// <summary>
202 /// <para>Ensures that the argument with the maximum value is greater than or equal to
203   ↳ the minimum value. This check is performed only for DEBUG build configuration.</para>
204 /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
205   ↳ минимальному значению. Эта проверка выполняется только для конфигурации сборки
206   ↳ DEBUG.</para>
207 /// </summary>
208 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
209   ↳ аргумента.</para></typeparam>
210 /// <param name="root"><para>The extension root to which this method is
211   ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
212 /// <param name="minimumArgument"><para>The argument with the minimum
213   ↳ value.</para><para>Аргумент с минимальным значением.</para></param>
214 /// <param name="maximumArgument"><para>The argument with the maximum
215   ↳ value.</para><para>Аргумент с максимальным значением.</para></param>
216 /// <param name="maximumArgumentName"><para>The name of argument with the maximum
217   ↳ value.</para><para>Имя аргумента с максимальным значением.</para></param>
218 /// <param name="message"><para>The message of the thrown
219   ↳ exception.</para><para>Сообщение выбрасываемого исключения.</para></param>
220 [Conditional("DEBUG")]
221 public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
222   ↳ EnsureOnDebugExtensionRoot root, TArgument minimumArgument, TArgument
223   ↳ maximumArgument, string maximumArgumentName, string message) =>
224   ↳ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimumArgument,
225   ↳ maximumArgument, maximumArgumentName, message);
226
227 /// <summary>
228 /// <para>Ensures that the argument with the maximum value is greater than or equal to
229   ↳ the minimum value. This check is performed only for DEBUG build configuration.</para>
230 /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
231   ↳ минимальному значению. Эта проверка выполняется только для конфигурации сборки
232   ↳ DEBUG.</para>
233 /// </summary>
234 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
235   ↳ аргумента.</para></typeparam>

```

```

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

```

```

264 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argument, Range<TArgument> range, string argumentName) =>
    ↳ Ensure.Always.ArgumentInRange(argument, range, argumentName);

265
266 /// <summary>
267 /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed only for DEBUG build configuration.</para>
268 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется только для конфигурации сборки DEBUG.</para>
269 /// </summary>
270 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
271 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
272 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
    ↳ аргумента.</para></param>
273 /// <param name="minimum"><para>The minimum possible argument's
    ↳ value.</para><para>Минимально возможное значение аргумента.</para></param>
274 /// <param name="maximum"><para>The maximum possible argument's
    ↳ value.</para><para>Максимально возможное значение аргумента.</para></param>
275 [Conditional("DEBUG")]
276 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argumentValue, TArgument minimum, TArgument maximum) =>
    ↳ Ensure.Always.ArgumentInRange(argumentValue, new Range<TArgument>(minimum, maximum),
    ↳ null);

277
278 /// <summary>
279 /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed only for DEBUG build configuration.</para>
280 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется только для конфигурации сборки DEBUG.</para>
281 /// </summary>
282 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
283 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
284 /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
    ↳ аргумента.</para></param>
285 /// <param name="minimum"><para>The minimum possible argument's
    ↳ value.</para><para>Минимально возможное значение аргумента.</para></param>
286 /// <param name="maximum"><para>The maximum possible argument's
    ↳ value.</para><para>Максимально возможное значение аргумента.</para></param>
287 /// <param name="argumentName"><para>The argument's name.</para><para>Имя
    ↳ аргумента.</para></param>
288 [Conditional("DEBUG")]
289 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argumentValue, TArgument minimum, TArgument maximum, string argumentName)
    ↳ => Ensure.Always.ArgumentInRange(argumentValue, new Range<TArgument>(minimum,
    ↳ maximum), argumentName);

290
291 /// <summary>
292 /// <para>Ensures that the argument value is in the specified range. This check is
    ↳ performed only for DEBUG build configuration.</para>
293 /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
    ↳ проверка выполняется только для конфигурации сборки DEBUG.</para>
294 /// </summary>
295 /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
    ↳ аргумента.</para></typeparam>
296 /// <param name="root"><para>The extension root to which this method is
    ↳ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
297 /// <param name="argument"></param>
298 /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
    ↳ диапазона.</para></param>
299 [Conditional("DEBUG")]
300 public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
    ↳ TArgument argument, Range<TArgument> range) =>
    ↳ Ensure.Always.ArgumentInRange(argument, range, null);

301
302 #endregion
303 }
304 }

```

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);

```



```
74     }
75 }
```

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

```
1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Ranges
4 {
5     /// <summary>
6     /// <para>Represents a set of extension methods for <see cref="Range{T}"/> structs.</para>
7     /// <para>Представляет 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>
```

```

50     /// </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
98     ↪ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
99     ↪ и <see cref="Range{T}.Maximum"/>.</para></returns>
100     [MethodImpl(MethodImplOptions.AggressiveInlining)]
101     public static float Difference(this Range<float> range) => range.Maximum - range.Minimum;
102
103     /// <summary>
104     /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see
105     ↪ cref="Range{T}.Maximum"/>.</para>
106     /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see
107     ↪ cref="Range{T}.Maximum"/>.</para>
108     /// </summary>
109     /// <param name="range"><para>The range of <see cref="decimal"/>.</para><para>Диапазон
110     ↪ значений <see cref="decimal"/>.</para></param>
111     /// <returns><para>Difference between <see cref="Range{T}.Minimum"/> and <see
112     ↪ cref="Range{T}.Maximum"/>.</para><para>Разницу между <see cref="Range{T}.Minimum"/>
113     ↪ и <see cref="Range{T}.Maximum"/>.</para></returns>
114     [MethodImpl(MethodImplOptions.AggressiveInlining)]
115     public static decimal Difference(this Range<decimal> range) => range.Maximum -
116     ↪ range.Minimum;
117 }
118 }

```

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 minimum 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_
14     ↪ pe-for-representing-an-integer-range">the question at
15     ↪ StackOverflow</a>.</para>
16     /// <para>Основано на <a href="http://stackoverflow.com/questions/5343006/is-there-a-c-sharp_
17     ↪ -type-for-representing-an-integer-range">вопросе в
18     ↪ StackOverflow</a>.</para>
19     /// </remarks>
20     public struct Range<T> : IEquatable<Range<T>>
21     {
22         /// <summary>
23         /// <para>
24         /// The default.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         private static readonly Comparer<T> _comparer = Comparer<T>.Default;
29         /// <summary>
30         /// <para>
31         /// The default.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         private static readonly EqualityComparer<T> _equalityComparer =
36         ↪ EqualityComparer<T>.Default;
37
38         /// <summary>
39         /// <para>Returns minimum value of the range.</para>
40         /// <para>Возвращает минимальное значение диапазона.</para>
41         /// </summary>
42         public readonly T Minimum;
43
44         /// <summary>
45         /// <para>Returns maximum value of the range.</para>
46         /// <para>Возвращает максимальное значение диапазона.</para>
47         /// </summary>
48         public readonly T Maximum;
49
50         /// <summary>
51         /// <para>Initializes a new instance of the Range class.</para>
52         /// <para>Инициализирует новый экземпляр класса Range.</para>
53         /// </summary>

```

```

49  /// <param name="minimumAndMaximum"><para>Single value for both Minimum and Maximum
    ↳ fields.</para><para>Одно значение для полей Minimum и Maximum.</para></param>
50  [MethodImpl(MethodImplOptions.AggressiveInlining)]
51  public Range(T minimumAndMaximum)
52  {
53      Minimum = minimumAndMaximum;
54      Maximum = minimumAndMaximum;
55  }
56
57  /// <summary>
58  /// <para>Initializes a new instance of the Range class.</para>
59  /// <para>Инициализирует новый экземпляр класса Range.</para>
60  /// </summary>
61  /// <param name="minimum"><para>The minimum value of the range.</para><para>Минимальное
    ↳ значение диапазона.</para></param>
62  /// <param name="maximum"><para>The maximum value of the range.</para><para>Максимальное
    ↳ значение диапазона.</para></param>
63  /// <exception cref="ArgumentException"><para>Thrown when the maximum is less than the
    ↳ minimum.</para><para>Выбрасывается, когда максимум меньше
    ↳ минимума.</para></exception>
64  [MethodImpl(MethodImplOptions.AggressiveInlining)]
65  public Range(T minimum, T maximum)
66  {
67      Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
    ↳ nameof(maximum));
68      Minimum = minimum;
69      Maximum = maximum;
70  }
71
72  /// <summary>
73  /// <para>Presents the Range in readable format.</para>
74  /// <para>Представляет диапазон в удобном для чтения формате.</para>
75  /// </summary>
76  /// <returns><para>String representation of the Range.</para><para>Строковое
    ↳ представление диапазона.</para></returns>
77  [MethodImpl(MethodImplOptions.AggressiveInlining)]
78  public override string ToString() => $"{Minimum}..{Maximum}";
79
80  /// <summary>
81  /// <para>Determines if the provided value is inside the range.</para>
82  /// <para>Определяет, находится ли указанное значение внутри диапазона.</para>
83  /// </summary>
84  /// <param name="value"><para>The value to test.</para><para>Значение для
    ↳ проверки.</para></param>
85  /// <returns><para>True if the value is inside Range, else false.</para><para>True, если
    ↳ значение находится внутри диапазона, иначе false.</para></returns>
86  [MethodImpl(MethodImplOptions.AggressiveInlining)]
87  public bool Contains(T value) => _comparer.Compare(Minimum, value) <= 0 &&
    ↳ _comparer.Compare(Maximum, value) >= 0;
88
89  /// <summary>
90  /// <para>Determines if another range is inside the bounds of this range.</para>
91  /// <para>Определяет, находится ли другой диапазон внутри границ этого диапазона.</para>
92  /// </summary>
93  /// <param name="range"><para>The child range to test.</para><para>Дочерний диапазон для
    ↳ проверки.</para></param>
94  /// <returns><para>True if range is inside, else false.</para><para>True, если диапазон
    ↳ находится внутри, иначе false.</para></returns>
95  [MethodImpl(MethodImplOptions.AggressiveInlining)]
96  public bool Contains(Range<T> range) => Contains(range.Minimum) &&
    ↳ Contains(range.Maximum);
97
98  /// <summary>
99  /// <para>Determines whether the current range is equal to another range.</para>
100  /// <para>Определяет, равен ли текущий диапазон другому диапазону.</para>
101  /// </summary>
102  /// <param name="other"><para>A range to compare with this range.</para><para>Диапазон
    ↳ для сравнения с этим диапазоном.</para></param>
103  /// <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>
104  [MethodImpl(MethodImplOptions.AggressiveInlining)]
105  public bool Equals(Range<T> other) => _equalityComparer.Equals(Minimum, other.Minimum)
    ↳ && _equalityComparer.Equals(Maximum, other.Maximum);
106
107  /// <summary>

```

```

108  /// <para>Creates a new <see cref="ValueTuple{T,T}"> struct initialized with <see
109  → cref="Range{T}.Minimum"/> as <see cref="ValueTuple{T,T}.Item1"/> and <see
110  → cref="Range{T}.Maximum"/> as <see cref="ValueTuple{T,T}.Item2"/>.</para>
111  /// <para>Создает новую структуру <see cref="ValueTuple{T,T}">, инициализированную с
112  → помощью <see cref="Range{T}.Minimum"/> как <see cref="ValueTuple{T,T}.Item1"/> и
113  → <see cref="Range{T}.Maximum"/> как <see cref="ValueTuple{T,T}.Item2"/>.</para>
114  /// </summary>
115  /// <param name="range"><para>The range of <typeparamref
116  → name="T"/>.</para><para>Диапазон значений <typeparamref name="T"/>.</para></param>
117  [MethodImpl(MethodImplOptions.AggressiveInlining)]
118  public static implicit operator ValueTuple<T, T>(Range<T> range) => (range.Minimum,
119  → range.Maximum);
120
121  /// <summary>
122  /// <para>Creates a new <see cref="Range{T}"> struct initialized with <see
123  → cref="ValueTuple{T,T}.Item1"/> as <see cref="Range{T}.Minimum"/> and <see
124  → cref="ValueTuple{T,T}.Item2"/> as <see cref="Range{T}.Maximum"/>.</para>
125  /// <para>Создает новую структуру <see cref="Range{T}">, инициализированную с помощью
126  → <see cref="ValueTuple{T,T}.Item1"/> как <see cref="Range{T}.Minimum"/> и <see
127  → cref="ValueTuple{T,T}.Item2"/> как <see cref="Range{T}.Maximum"/>.</para>
128  /// </summary>
129  /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
130  [MethodImpl(MethodImplOptions.AggressiveInlining)]
131  public static implicit operator Range<T>(ValueTuple<T, T> tuple) => new
132  → Range<T>(tuple.Item1, tuple.Item2);
133
134  /// <summary>
135  /// <para>Determines whether the current range is equal to another object.</para>
136  /// <para>Определяет, равен ли текущий диапазон другому объекту.</para>
137  /// </summary>
138  /// <param name="obj"><para>An object to compare with this range.</para><para>Объект для
139  → сравнения с этим диапазоном.</para></param>
140  /// <returns><para>True if the current range is equal to the other object; otherwise,
141  → false.</para><para>True, если текущий диапазон равен другому объекту; иначе
142  → false.</para></returns>
143  [MethodImpl(MethodImplOptions.AggressiveInlining)]
144  public override bool Equals(object obj) => obj is Range<T> range ? Equals(range) : false;
145
146  /// <summary>
147  /// Calculates the hash code for the current <see cref="Range{T}"> instance.
148  /// </summary>
149  /// <returns>The hash code for the current <see cref="Range{T}"> instance.</returns>
150  [MethodImpl(MethodImplOptions.AggressiveInlining)]
151  public override int GetHashCode() => (Minimum, Maximum).GetHashCode();
152
153  /// <summary>
154  /// <para>Determines if the specified range is equal to the current range.</para>
155  /// <para>Определяет, равен ли указанный диапазон текущему диапазону.</para>
156  /// </summary>
157  /// <param name="left"><para>The current range.</para><para>Текущий
158  → диапазон.</para></param>
159  /// <param name="right"><para>A range to compare with this range.</para><para>Диапазон
160  → для сравнения с этим диапазоном.</para></param>
161  /// <returns><para>True if the current range is equal to the other range; otherwise,
162  → false.</para><para>True, если текущий диапазон равен другому диапазону; иначе
163  → false.</para></returns>
164  [MethodImpl(MethodImplOptions.AggressiveInlining)]
165  public static bool operator ==(Range<T> left, Range<T> right) => left.Equals(right);
166
167  /// <summary>
168  /// <para>Determines if the specified range is not equal to the current range.</para>
169  /// <para>Определяет, не равен ли указанный диапазон текущему диапазону.</para>
170  /// </summary>
171  /// <param name="left"><para>The current range.</para><para>Текущий
172  → диапазон.</para></param>
173  /// <param name="right"><para>A range to compare with this range.</para><para>Диапазон
174  → для сравнения с этим диапазоном.</para></param>
175  /// <returns><para>True if the current range is not equal to the other range; otherwise,
176  → false.</para><para>True, если текущий диапазон не равен другому диапазону; иначе
177  → false.</para></returns>
178  [MethodImpl(MethodImplOptions.AggressiveInlining)]
179  public static bool operator !=(Range<T> left, Range<T> right) => !(left == right);
180
181  }

```

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     /// <summary>
8     /// <para>
9     /// Represents the ensure extensions tests.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public static class EnsureExtensionsTests
14    {
15        /// <summary>
16        /// <para>
17        /// Tests that maximum argument is greater or equal to minimum exception test.
18        /// </para>
19        /// <para></para>
20        /// </summary>
21        [Fact]
22        public static void MaximumArgumentIsGreaterOrEqualToMinimumExceptionTest() =>
23            Assert.Throws<ArgumentException>(() =>
24                Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(2, 1));
25
26        /// <summary>
27        /// <para>
28        /// Tests that argument in range exception test.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        [Fact]
33        public static void ArgumentInRangeExceptionTest() =>
34            Assert.Throws<ArgumentOutOfRangeException>(() => Ensure.Always.ArgumentInRange(5,
35                (6, 7)));
36    }
37 }
```

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

```
1 using System;
2 using Xunit;
3
4 namespace Platform.Ranges.Tests
5 {
6     /// <summary>
7     /// <para>
8     /// Represents the range tests.
9     /// </para>
10    /// <para></para>
11    /// </summary>
12    public static class RangeTests
13    {
14        /// <summary>
15        /// <para>
16        /// Tests that constructors test.
17        /// </para>
18        /// <para></para>
19        /// </summary>
20        [Fact]
21        public static void ConstructorsTest()
22        {
23            var range1 = new Range<int>(1, 3);
24            Assert.Equal(1, range1.Minimum);
25            Assert.Equal(3, range1.Maximum);
26            Assert.Throws<ArgumentException>(() => new Range<int>(2, 1));
27            var range2 = new Range<int>(5);
28            Assert.Equal(5, range2.Minimum);
29            Assert.Equal(5, range2.Maximum);
30        }
31
32        /// <summary>
33        /// <para>
34        /// Tests that contains test.
35        /// </para>
36        /// <para></para>
37        /// </summary>
38        [Fact]
```

```

39 public static void ContainsTest()
40 {
41     var range = new Range<int>(1, 3);
42     Assert.True(range.Contains(1));
43     Assert.True(range.Contains(2));
44     Assert.True(range.Contains(3));
45     Assert.True(range.Contains((2, 3)));
46     Assert.False(range.Contains((3, 4)));
47 }
48
49 /// <summary>
50 /// <para>
51 /// Tests that difference test.
52 /// </para>
53 /// <para></para>
54 /// </summary>
55 [Fact]
56 public static void DifferenceTest()
57 {
58     var range = new Range<int>(1, 3);
59     Assert.Equal(2, range.Difference());
60 }
61
62 /// <summary>
63 /// <para>
64 /// Tests that to string test.
65 /// </para>
66 /// <para></para>
67 /// </summary>
68 [Fact]
69 public static void ToStringTest()
70 {
71     var range = new Range<int>(1, 3);
72     Assert.Equal("[1..3]", range.ToString());
73 }
74
75 /// <summary>
76 /// <para>
77 /// Tests that equality test.
78 /// </para>
79 /// <para></para>
80 /// </summary>
81 [Fact]
82 public static void EqualityTest()
83 {
84     var range1 = new Range<int>(1, 3);
85     var range1Duplicate = new Range<int>(1, 3);
86     var range2 = new Range<int>(2, 5);
87     Assert.True(range1 == range1Duplicate);
88     Assert.Equal(range1, range1Duplicate);
89     Assert.True(range1 != range2);
90     Assert.NotEqual(range1, range2);
91 }
92 }
93 }

```


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, 9
- ./csharp/Platform.Ranges/Range[T].cs, 11