1.1 ./EnsureExtensions.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Runtime.CompilerServices;
using Platform.Exceptions;
using Platform.Exceptions.ExtensionRoots;

#pragma warning disable IDE0060 // Remove unused parameter

namespace Platform.Ranges
{
    /// <summary>
    /// <para>Provides a set of extension methods for <see cref="EnsureAlwaysExtensionRoot"/>
    ↪  and <see cref="EnsureOnDebugExtensionRoot"/> objects.</para>
    /// <para>Предоставляет набор методов расширения для объектов <see
    ↪  cref="EnsureAlwaysExtensionRoot"/> и <see cref="EnsureOnDebugExtensionRoot"/>.</para>
    /// </summary>
    public static class EnsureExtensions
    {
        private const string DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage = "Maximum
        ↪  should be greater or equal to minimum.";

        #region Always

        /// <summary>
        /// <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>
        /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
        ↪  минимальному значению. Эта проверка выполняется внезависимости от конфигурации
        ↪  сборки.</para>
        /// </summary>
        /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
        ↪  аргумента.</para></typeparam>
        /// <param name="root"><para>The extension root to which this method is
        ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
        /// <param name="minimumArgument"><para>The argument with the minimum
        ↪  value.</para><para>Аргумент с минимальным значением.</para></param>
        /// <param name="maximumArgument"><para>The argument with the maximum
        ↪  value.</para><para>Аргумент с максимальным значением.</para></param>
        /// <param name="maximumArgumentName"><para>The name of argument with the maximum
        ↪  value.</para><para>Имя аргумента с максимальным значением.</para></param>
        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void MaximumArgumentIsGreaterOrEqualToMinimum<TArgument>(this
        ↪  EnsureAlwaysExtensionRoot root, TArgument minimumArgument, TArgument
        ↪  maximumArgument, string maximumArgumentName, Func<string> messageBuilder)
        {
            if (Comparer<TArgument>.Default.Compare(maximumArgument, minimumArgument) < 0)
            {
                throw new ArgumentException(messageBuilder(), maximumArgumentName);
            }
        }

        /// <summary>
        /// <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>
        /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
        ↪  минимальному значению. Эта проверка выполняется внезависимости от конфигурации
        ↪  сборки.</para>
        /// </summary>
        /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
        ↪  аргумента.</para></typeparam>
        /// <param name="root"><para>The extension root to which this method is
        ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
        /// <param name="minimumArgument"><para>The argument with the minimum
        ↪  value.</para><para>Аргумент с минимальным значением.</para></param>
        /// <param name="maximumArgument"><para>The argument with the maximum
        ↪  value.</para><para>Аргумент с максимальным значением.</para></param>
        /// <param name="maximumArgumentName"><para>The name of argument with the maximum
        ↪  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 внезависимости 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 внезависимости 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 внезависимости 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 внезависимости 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 внезависимости 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 внезависимости 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
            ↪ аргумента.</para></param>
140         /// <param name="minimum"><para>The minimum possible argument's
            ↪ value.</para><para>Минимально возможное значение аргумента.</para></param>
141         /// <param name="maximum"><para>The maximum possible argument's
            ↪ value.</para><para>Максимально возможное значение аргумента.</para></param>
142         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪ аргумента.</para></param>
143         [MethodImpl(MethodImplOptions.AggressiveInlining)]
144         public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪ TArgument argumentValue, TArgument minimum, TArgument maximum, string argumentName)
            ↪ => ArgumentInRange(root, argumentValue, new Range<TArgument>(minimum, maximum),
            ↪ argumentName);
145
146         /// <summary>
147         /// <para>Ensures that the argument value is in the specified range. This check is
            ↪ performed regardless of the build configuration.</para>
148         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪ проверка выполняется внезависимости от конфигурации сборки.</para>
149         /// </summary>
150         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪ аргумента.</para></typeparam>
151         /// <param name="root"><para>The extension root to which this method is
            ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
152         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪ аргумента.</para></param>
153         /// <param name="minimum"><para>The minimum possible argument's
            ↪ value.</para><para>Минимально возможное значение аргумента.</para></param>
154         /// <param name="maximum"><para>The maximum possible argument's
            ↪ value.</para><para>Максимально возможное значение аргумента.</para></param>
155         [MethodImpl(MethodImplOptions.AggressiveInlining)]
156         public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪ TArgument argumentValue, TArgument minimum, TArgument maximum) =>
            ↪ ArgumentInRange(root, argumentValue, new Range<TArgument>(minimum, maximum), null);
157
158         /// <summary>
159         /// <para>Ensures that the argument value is in the specified range. This check is
            ↪ performed regardless of the build configuration.</para>
160         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪ проверка выполняется внезависимости от конфигурации сборки.</para>
161         /// </summary>
162         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪ аргумента.</para></typeparam>
163         /// <param name="root"><para>The extension root to which this method is
            ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
164         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪ аргумента.</para></param>
165         /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
            ↪ диапазона.</para></param>
166         [MethodImpl(MethodImplOptions.AggressiveInlining)]
167         public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪ TArgument argumentValue, Range<TArgument> range) => ArgumentInRange(root,
            ↪ argumentValue, range, null);
168
169         #endregion
170
171         #region OnDebug
172
173         /// <summary>
174         /// <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>
175         /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
            ↪ минимальному значению. Эта проверка выполняется только для конфигурации сборки
            ↪ DEBUG.</para>
176         /// </summary>
177         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪ аргумента.</para></typeparam>
178         /// <param name="root"><para>The extension root to which this method is
            ↪ bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
179         /// <param name="minimumArgument"><para>The argument with the minimum
            ↪ value.</para><para>Аргумент с минимальным значением.</para></param>
180         /// <param name="maximumArgument"><para>The argument with the maximum
            ↪ value.</para><para>Аргумент с максимальным значением.</para></param>
181         /// <param name="maximumArgumentName"><para>The name of argument with the maximum
            ↪ 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 сообщение 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
            ↪  performed only for DEBUG build configuration.</para>
262         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪  проверка выполняется только для конфигурации сборки DEBUG.</para>
263         /// </summary>
264         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
265         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
266         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪  аргумента.</para></param>
267         /// <param name="minimum"><para>The minimum possible argument's
            ↪  value.</para><para>Минимально возможное значение аргумента.</para></param>
268         /// <param name="maximum"><para>The maximum possible argument's
            ↪  value.</para><para>Максимально возможное значение аргумента.</para></param>
269         [Conditional("DEBUG")]
270         public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
            ↪  TArgument argumentValue, TArgument minimum, TArgument maximum) =>
            ↪  Ensure.Always.ArgumentInRange(argumentValue, new Range<TArgument>(minimum, maximum),
            ↪  null);
271
272         /// <summary>
273         /// <para>Ensures that the argument value is in the specified range. This check is
            ↪  performed only for DEBUG build configuration.</para>
274         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪  проверка выполняется только для конфигурации сборки DEBUG.</para>
275         /// </summary>
276         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
277         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
278         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪  аргумента.</para></param>
279         /// <param name="minimum"><para>The minimum possible argument's
            ↪  value.</para><para>Минимально возможное значение аргумента.</para></param>
280         /// <param name="maximum"><para>The maximum possible argument's
            ↪  value.</para><para>Максимально возможное значение аргумента.</para></param>
281         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
282         [Conditional("DEBUG")]
283         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);
284
285         /// <summary>
286         /// <para>Ensures that the argument value is in the specified range. This check is
            ↪  performed only for DEBUG build configuration.</para>
287         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪  проверка выполняется только для конфигурации сборки DEBUG.</para>
288         /// </summary>
289         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
290         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
291         /// <param name="argument"></param>
292         /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
            ↪  диапазона.</para></param>
293         [Conditional("DEBUG")]
294         public static void ArgumentInRange<TArgument>(this EnsureOnDebugExtensionRoot root,
            ↪  TArgument argument, Range<TArgument> range) =>
            ↪  Ensure.Always.ArgumentInRange(argument, range, null);
295
296         #endregion
297     }
298 }
```

1.2 ./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,
            ↪  sbyte.MaxValue);
14
15          /// <summary>
16          /// <para>Gets the whole <see cref="short"/> values range.</para>
17          /// <para>Возвращает весь диапазон значений <see cref="short"/>.</para>
18          /// </summary>
19          public static readonly Range<short> Int16 = new Range<short>(short.MinValue,
            ↪  short.MaxValue);
20
21          /// <summary>
22          /// <para>Gets the whole <see cref="int"/> values range.</para>
23          /// <para>Возвращает весь диапазон значений <see cref="int"/>.</para>
24          /// </summary>
25          public static readonly Range<int> Int32 = new Range<int>(int.MinValue, int.MaxValue);
26
27          /// <summary>
28          /// <para>Gets the whole <see cref="long"/> values range.</para>
29          /// <para>Возвращает весь диапазон значений <see cref="long"/>.</para>
30          /// </summary>
31          public static readonly Range<long> Int64 = new Range<long>(long.MinValue, long.MaxValue);
32
33          /// <summary>
34          /// <para>Gets the whole <see cref="byte"/> values range.</para>
35          /// <para>Возвращает весь диапазон значений <see cref="byte"/>.</para>
36          /// </summary>
37          public static readonly Range<byte> Byte = new Range<byte>(byte.MinValue, byte.MaxValue);
38
39          /// <summary>
40          /// <para>Gets the whole <see cref="ushort"/> values range.</para>
41          /// <para>Возвращает весь диапазон значений <see cref="ushort"/>.</para>
42          /// </summary>
43          public static readonly Range<ushort> UInt16 = new Range<ushort>(ushort.MinValue,
            ↪  ushort.MaxValue);
44
45          /// <summary>
46          /// <para>Gets the whole <see cref="uint"/> values range.</para>
47          /// <para>Возвращает весь диапазон значений <see cref="uint"/>.</para>
48          /// </summary>
49          public static readonly Range<uint> UInt32 = new Range<uint>(uint.MinValue,
            ↪  uint.MaxValue);
50
51          /// <summary>
52          /// <para>Gets the whole <see cref="ulong"/> values range.</para>
53          /// <para>Возвращает весь диапазон значений <see cref="ulong"/>.</para>
54          /// </summary>
55          public static readonly Range<ulong> UInt64 = new Range<ulong>(ulong.MinValue,
            ↪  ulong.MaxValue);
56
57          /// <summary>
58          /// <para>Gets the whole <see cref="float"/> values range.</para>
59          /// <para>Возвращает весь диапазон значений <see cref="float"/>.</para>
60          /// </summary>
61          public static readonly Range<float> Single = new Range<float>(float.MinValue,
            ↪  float.MaxValue);
62
63          /// <summary>
64          /// <para>Gets the whole <see cref="double"/> values range.</para>
65          /// <para>Возвращает весь диапазон значений <see cref="double"/>.</para>
66          /// </summary>
67          public static readonly Range<double> Double = new Range<double>(double.MinValue,
            ↪  double.MaxValue);
68
69          /// <summary>
70          /// <para>Gets the whole <see cref="decimal"/> values range.</para>
71          /// <para>Возвращает весь диапазон значений <see cref="decimal"/>.</para>
72          /// </summary>
73          public static readonly Range<decimal> Decimal = new Range<decimal>(decimal.MinValue,
            ↪  decimal.MaxValue);
74      }
75  }
```

1.3 ./RangeExtensions.cs

```
using System.Runtime.CompilerServices;

namespace Platform.Ranges
{
    /// <summary>
    /// <para>Provides a set of extension methods for <see cref="Range{T}"/> structs.</para>
    /// <para>Предоставляет набор методов расширения для структур <see cref="Range{T}"/>.</para>
    /// </summary>
    public static class RangeExtensions
    {
        /// <summary>
        /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see cref="Range{T}.Maximum"/>.</para>
        /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see cref="Range{T}.Maximum"/>.</para>
        /// </summary>
        /// <param name="range"><para>The range of <see cref="ulong"/>.</para><para>Диапазон значений <see cref="ulong"/>.</para></param>
        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong Difference(this Range<ulong> range) => range.Maximum - range.Minimum;

        /// <summary>
        /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see cref="Range{T}.Maximum"/>.</para>
        /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see cref="Range{T}.Maximum"/>.</para>
        /// </summary>
        /// <param name="range"><para>The range of <see cref="uint"/>.</para><para>Диапазон значений <see cref="uint"/>.</para></param>
        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static uint Difference(this Range<uint> range) => range.Maximum - range.Minimum;

        /// <summary>
        /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see cref="Range{T}.Maximum"/>.</para>
        /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see cref="Range{T}.Maximum"/>.</para>
        /// </summary>
        /// <param name="range"><para>The range of <see cref="ushort"/>.</para><para>Диапазон значений <see cref="ushort"/>.</para></param>
        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ushort Difference(this Range<ushort> range) => (ushort)(range.Maximum - range.Minimum);

        /// <summary>
        /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see cref="Range{T}.Maximum"/>.</para>
        /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see cref="Range{T}.Maximum"/>.</para>
        /// </summary>
        /// <param name="range"><para>The range of <see cref="byte"/>.</para><para>Диапазон значений <see cref="byte"/>.</para></param>
        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static byte Difference(this Range<byte> range) => (byte)(range.Maximum - range.Minimum);

        /// <summary>
        /// <para>Calculates difference between <see cref="Range{T}.Minimum"/> and <see cref="Range{T}.Maximum"/>.</para>
        /// <para>Вычисляет разницу между <see cref="Range{T}.Minimum"/> и <see cref="Range{T}.Maximum"/>.</para>
        /// </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 ./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 }
```

Index