1.1 ./csharp/Platform.Ranges/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
    {
        /// <summary>
        /// <para>
        /// The default maximum should be greater or equal to minimum message.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
```

```
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 внезависимости 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 внезависимости 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 внезависимости 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 сообщение 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 внезависимости 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 внезависимости 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
            ↪  проверка выполняется внезависимости от конфигурации сборки.</para>
142         /// </summary>
143         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
144         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
145         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪  аргумента.</para></param>
146         /// <param name="minimum"><para>The minimum possible argument's
            ↪  value.</para><para>Минимально возможное значение аргумента.</para></param>
147         /// <param name="maximum"><para>The maximum possible argument's
            ↪  value.</para><para>Максимально возможное значение аргумента.</para></param>
148         /// <param name="argumentName"><para>The argument's name.</para><para>Имя
            ↪  аргумента.</para></param>
149         [MethodImpl(MethodImplOptions.AggressiveInlining)]
150         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);
151
152         /// <summary>
153         /// <para>Ensures that the argument value is in the specified range. This check is
            ↪  performed regardless of the build configuration.</para>
154         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪  проверка выполняется внезависимости от конфигурации сборки.</para>
155         /// </summary>
156         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
157         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
158         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪  аргумента.</para></param>
159         /// <param name="minimum"><para>The minimum possible argument's
            ↪  value.</para><para>Минимально возможное значение аргумента.</para></param>
160         /// <param name="maximum"><para>The maximum possible argument's
            ↪  value.</para><para>Максимально возможное значение аргумента.</para></param>
161         [MethodImpl(MethodImplOptions.AggressiveInlining)]
162         public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argumentValue, TArgument minimum, TArgument maximum) =>
            ↪  ArgumentInRange(root, argumentValue, new Range<TArgument>(minimum, maximum), null);
163
164         /// <summary>
165         /// <para>Ensures that the argument value is in the specified range. This check is
            ↪  performed regardless of the build configuration.</para>
166         /// <para>Гарантирует, что значение аргумента находится в указанном диапазоне. Эта
            ↪  проверка выполняется внезависимости от конфигурации сборки.</para>
167         /// </summary>
168         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
169         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
170         /// <param name="argumentValue"><para>The argument's value.</para><para>Значение
            ↪  аргумента.</para></param>
171         /// <param name="range"><para>The range restriction.</para><para>Ограничение в виде
            ↪  диапазона.</para></param>
172         [MethodImpl(MethodImplOptions.AggressiveInlining)]
173         public static void ArgumentInRange<TArgument>(this EnsureAlwaysExtensionRoot root,
            ↪  TArgument argumentValue, Range<TArgument> range) => ArgumentInRange(root,
            ↪  argumentValue, range, null);
174
175         #endregion
176
177         #region OnDebug
178
179         /// <summary>
180         /// <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>
181         /// <para>Гарантирует, что аргумент с максимальным значением больше или равен
            ↪  минимальному значению. Эта проверка выполняется только для конфигурации сборки
            ↪  DEBUG.</para>
182         /// </summary>
183         /// <typeparam name="TArgument"><para>Type of argument.</para><para>Тип
            ↪  аргумента.</para></typeparam>
184         /// <param name="root"><para>The extension root to which this method is
            ↪  bound.</para><para>Корень-расширения, к которому привязан этот метод.</para></param>
```

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

```
    /// <para>Contains static fields with <see cref="Range{T}"/> constants.</para>
    /// <para>Содержит статические поля с константами типа <see cref="Range{T}"/>.</para>
    /// </summary>
    public static class Range
    {
        /// <summary>
        /// <para>Gets the whole <see cref="sbyte"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="sbyte"/>.</para>
        /// </summary>
        public static readonly Range<sbyte> SByte = new Range<sbyte>(sbyte.MinValue,
        ↪  sbyte.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="short"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="short"/>.</para>
        /// </summary>
        public static readonly Range<short> Int16 = new Range<short>(short.MinValue,
        ↪  short.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="int"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="int"/>.</para>
        /// </summary>
        public static readonly Range<int> Int32 = new Range<int>(int.MinValue, int.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="long"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="long"/>.</para>
        /// </summary>
        public static readonly Range<long> Int64 = new Range<long>(long.MinValue, long.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="byte"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="byte"/>.</para>
        /// </summary>
        public static readonly Range<byte> Byte = new Range<byte>(byte.MinValue, byte.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="ushort"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="ushort"/>.</para>
        /// </summary>
        public static readonly Range<ushort> UInt16 = new Range<ushort>(ushort.MinValue,
        ↪  ushort.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="uint"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="uint"/>.</para>
        /// </summary>
        public static readonly Range<uint> UInt32 = new Range<uint>(uint.MinValue,
        ↪  uint.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="ulong"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="ulong"/>.</para>
        /// </summary>
        public static readonly Range<ulong> UInt64 = new Range<ulong>(ulong.MinValue,
        ↪  ulong.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="float"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="float"/>.</para>
        /// </summary>
        public static readonly Range<float> Single = new Range<float>(float.MinValue,
        ↪  float.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="double"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="double"/>.</para>
        /// </summary>
        public static readonly Range<double> Double = new Range<double>(double.MinValue,
        ↪  double.MaxValue);

        /// <summary>
        /// <para>Gets the whole <see cref="decimal"/> values range.</para>
        /// <para>Возвращает весь диапазон значений <see cref="decimal"/>.</para>
        /// </summary>
        public static readonly Range<decimal> Decimal = new Range<decimal>(decimal.MinValue,
        ↪  decimal.MaxValue);
```

```
    }
}
```

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

```
using System.Runtime.CompilerServices;

namespace Platform.Ranges
{
    /// <summary>
    /// <para>Represents a set of extension methods for <see cref="Range{T}"/> structs.</para>
    /// <para>Представляет 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>
```

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

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

```
1   using System;
2   using System.Collections.Generic;
3   using System.Runtime.CompilerServices;
4   using Platform.Exceptions;
5
6   namespace Platform.Ranges
7   {
8       /// <summary>
9       /// <para>Represents a range between 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
            ↪  pe-for-representing-an-integer-range">the question at
            ↪  StackOverflow</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
            ↪  StackOverflow</a>.</para>
15      /// </remarks>
16      public struct Range<T> : IEquatable<Range<T>>
17      {
18          /// <summary>
19          /// <para>
20          /// The default.
21          /// </para>
22          /// <para></para>
23          /// </summary>
24          private static readonly Comparer<T> _comparer = Comparer<T>.Default;
25          /// <summary>
26          /// <para>
27          /// The default.
28          /// </para>
29          /// <para></para>
30          /// </summary>
31          private static readonly EqualityComparer<T> _equalityComparer =
            ↪  EqualityComparer<T>.Default;
32
33          /// <summary>
34          /// <para>Returns minimum value of the range.</para>
35          /// <para>Возвращает минимальное значение диапазона.</para>
36          /// </summary>
37          public readonly T Minimum;
38
39          /// <summary>
40          /// <para>Returns maximum value of the range.</para>
41          /// <para>Возвращает максимальное значение диапазона.</para>
42          /// </summary>
43          public readonly T Maximum;
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="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
            ↪  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>
109         /// <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>
110         /// </summary>
111         /// <param name="range"><para>The range of <typeparamref
            ↪  name="T"/>.</para><para>Диапазон значений <typeparamref name="T"/>.</para></param>
112         [MethodImpl(MethodImplOptions.AggressiveInlining)]
113         public static implicit operator ValueTuple<T, T>(Range<T> range) => (range.Minimum,
            ↪  range.Maximum);
114
115         /// <summary>
116         /// <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>
117         /// <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>
118         /// </summary>
119         /// <param name="tuple"><para>The tuple.</para><para>Кортеж.</para></param>
120         [MethodImpl(MethodImplOptions.AggressiveInlining)]
121         public static implicit operator Range<T>(ValueTuple<T, T> tuple) => new
            ↪  Range<T>(tuple.Item1, tuple.Item2);
122
123         /// <summary>
124         /// <para>Determines whether the current range is equal to another object.</para>
125         /// <para>Определяет, равен ли текущий диапазон другому объекту.</para>
126         /// </summary>
127         /// <param name="obj"><para>An object to compare with this range.</para><para>Объект для
            ↪  сравнения с этим диапазоном.</para></param>
128         /// <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>
129         [MethodImpl(MethodImplOptions.AggressiveInlining)]
130         public override bool Equals(object obj) => obj is Range<T> range ? Equals(range) : false;
131
132         /// <summary>
133         /// Calculates the hash code for the current <see cref="Range{T}"/> instance.
134         /// </summary>
135         /// <returns>The hash code for the current <see cref="Range{T}"/> instance.</returns>
136         [MethodImpl(MethodImplOptions.AggressiveInlining)]
137         public override int GetHashCode() => (Minimum, Maximum).GetHashCode();
138
139         /// <summary>
140         /// <para>Determines if the specified range is equal to the current range.</para>
141         /// <para>Определяет, равен ли указанный диапазон текущему диапазону.</para>
142         /// </summary>
143         /// <param name="left"><para>The current range.</para><para>Текущий
            ↪  диапазон.</para></param>
144         /// <param name="right"><para>A range to compare with this range.</para><para>Диапазон
            ↪  для сравнения с этим диапазоном.</para></param>
145         /// <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>
146         [MethodImpl(MethodImplOptions.AggressiveInlining)]
147         public static bool operator ==(Range<T> left, Range<T> right) => left.Equals(right);
148
149         /// <summary>
150         /// <para>Determines if the specified range is not equal to the current range.</para>
151         /// <para>Определяет, не равен ли указанный диапазон текущему диапазону.</para>
152         /// </summary>
153         /// <param name="left"><para>The current range.</para><para>Текущий
            ↪  диапазон.</para></param>
154         /// <param name="right"><para>A range to compare with this range.</para><para>Диапазон
            ↪  для сравнения с этим диапазоном.</para></param>
155         /// <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>
156         [MethodImpl(MethodImplOptions.AggressiveInlining)]
157         public static bool operator !=(Range<T> left, Range<T> right) => !(left == right);
158     }
159 }
```

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

```
using System;
using Xunit;
using Platform.Exceptions;

namespace Platform.Ranges.Tests
{
    /// <summary>
    /// <para>
    /// Represents the ensure extensions tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class EnsureExtensionsTests
    {
        /// <summary>
        /// <para>
        /// Tests that maximum argument is greater or equal to minimum exception test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void MaximumArgumentIsGreaterOrEqualToMinimumExceptionTest() =>
        ↪  Assert.Throws<ArgumentException>(() =>
        ↪  Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(2, 1));

        /// <summary>
        /// <para>
        /// Tests that argument in range exception test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void ArgumentInRangeExceptionTest() =>
        ↪  Assert.Throws<ArgumentOutOfRangeException>(() => Ensure.Always.ArgumentInRange(5,
        ↪  (6, 7)));
    }
}
```

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

```
using System;
using Xunit;

namespace Platform.Ranges.Tests
{
    /// <summary>
    /// <para>
    /// Represents the range tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class RangeTests
    {
        /// <summary>
        /// <para>
        /// Tests that constructors test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void ConstructorsTest()
        {
            var range1 = new Range<int>(1, 3);
            Assert.Equal(1, range1.Minimum);
            Assert.Equal(3, range1.Maximum);
            Assert.Throws<ArgumentException>(() => new Range<int>(2, 1));
            var range2 = new Range<int>(5);
            Assert.Equal(5, range2.Minimum);
            Assert.Equal(5, range2.Maximum);
        }

        /// <summary>
        /// <para>
        /// Tests that contains test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
```

```
        public static void ContainsTest()
        {
            var range = new Range<int>(1, 3);
            Assert.True(range.Contains(1));
            Assert.True(range.Contains(2));
            Assert.True(range.Contains(3));
            Assert.True(range.Contains((2, 3)));
            Assert.False(range.Contains((3, 4)));
        }

        /// <summary>
        /// <para>
        /// Tests that difference test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void DifferenceTest()
        {
            var range = new Range<int>(1, 3);
            Assert.Equal(2, range.Difference());
        }

        /// <summary>
        /// <para>
        /// Tests that to string test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void ToStringTest()
        {
            var range = new Range<int>(1, 3);
            Assert.Equal("[1..3]", range.ToString());
        }

        /// <summary>
        /// <para>
        /// Tests that equality test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void EqualityTest()
        {
            var range1 = new Range<int>(1, 3);
            var range1Duplicate = new Range<int>(1, 3);
            var range2 = new Range<int>(2, 5);
            Assert.True(range1 == range1Duplicate);
            Assert.Equal(range1, range1Duplicate);
            Assert.True(range1 != range2);
            Assert.NotEqual(range1, range2);
        }
    }
}
```

Index