

LinksPlatform's Platform.Ranges Class Library

./EnsureExtensions.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Runtime.CompilerServices;
5  using Platform.Exceptions;
6  using Platform.Exceptions.ExtensionRoots;
7
8  #pragma warning disable IDE0060 // Remove unused parameter
9
10 namespace Platform.Ranges
11 {
12     public static class EnsureExtensions
13     {
14         private const string DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage = "Maximum
            ↳ should be greater or equal to minimum.";
15
16         #region Always
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↳ EnsureAlwaysExtensionRoot root, T minimum, T maximum) =>
            ↳ MaximumArgumentIsGreaterOrEqualToMinimum(root, minimum, maximum, nameof(maximum));
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↳ EnsureAlwaysExtensionRoot root, T minimum, T maximum, string argumentName) =>
            ↳ MaximumArgumentIsGreaterOrEqualToMinimum(root, minimum, maximum, nameof(maximum),
            ↳ DefaultMaximumShouldBeGreaterOrEqualToMinimumMessage);
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↳ EnsureAlwaysExtensionRoot root, T minimum, T maximum, string argumentName, string
            ↳ message)
26         {
27             string messageBuilder() => message;
28             MaximumArgumentIsGreaterOrEqualToMinimum(root, minimum, maximum, argumentName,
            ↳ messageBuilder());
29         }
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
            ↳ EnsureAlwaysExtensionRoot root, T minimum, T maximum, string argumentName,
            ↳ Func<string> messageBuilder)
33         {
34             if (Comparer<T>.Default.Compare(maximum, minimum) < 0)
35             {
36                 throw new ArgumentException(messageBuilder(), argumentName);
37             }
38         }
39
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
            ↳ argumentValue, T minimum, T maximum) => ArgumentInRange(root, argumentValue, new
            ↳ Range<T>(minimum, maximum), null);
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
            ↳ argumentValue, T minimum, T maximum, string argumentName) => ArgumentInRange(root,
            ↳ argumentValue, new Range<T>(minimum, maximum), argumentName);
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
            ↳ argumentValue, Range<T> range) => ArgumentInRange(root, argumentValue, range, null);
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
            ↳ argumentValue, Range<T> range, string argumentName)
51         {
52             string messageBuilder() => $"Argument value [{argumentValue}] is out of range
            ↳ {range}.";
53             ArgumentInRange(root, argumentValue, range, argumentName, messageBuilder());
54         }
55
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
            ↳ argumentValue, Range<T> range, string argumentName, string message)

```

```

58 {
59     string messageBuilder() => message;
60     ArgumentInRange(root, argumentValue, range, argumentName, messageBuilder);
61 }
62
63 [MethodImpl(MethodImplOptions.AggressiveInlining)]
64 public static void ArgumentInRange<T>(this EnsureAlwaysExtensionRoot root, T
65     ↪ argumentValue, Range<T> range, string argumentName, Func<string> messageBuilder)
66 {
67     if (!range.ContainsValue(argumentValue))
68     {
69         throw new ArgumentOutOfRangeException(argumentName, argumentValue,
70             ↪ messageBuilder());
71     }
72 }
73
74 #endregion
75
76 #region OnDebug
77
78 [Conditional("DEBUG")]
79 public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
80     ↪ EnsureOnDebugExtensionRoot root, T minimum, T maximum) =>
81     ↪ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum, null);
82
83 [Conditional("DEBUG")]
84 public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
85     ↪ EnsureOnDebugExtensionRoot root, T minimum, T maximum, string argumentName) =>
86     ↪ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
87     ↪ argumentName);
88
89 [Conditional("DEBUG")]
90 public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
91     ↪ EnsureOnDebugExtensionRoot root, T minimum, T maximum, string argumentName, string
92     ↪ message) => Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
93     ↪ argumentName, message);
94
95 [Conditional("DEBUG")]
96 public static void MaximumArgumentIsGreaterOrEqualToMinimum<T>(this
97     ↪ EnsureOnDebugExtensionRoot root, T minimum, T maximum, string argumentName,
98     ↪ Func<string> messageBuilder) =>
99     ↪ Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
100     ↪ argumentName, messageBuilder);
101
102 [Conditional("DEBUG")]
103 public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T
104     ↪ argumentValue, T minimum, T maximum) => Ensure.Always.ArgumentInRange(argumentValue,
105     ↪ new Range<T>(minimum, maximum), null);
106
107 [Conditional("DEBUG")]
108 public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T
109     ↪ argumentValue, T minimum, T maximum, string argumentName) =>
110     ↪ Ensure.Always.ArgumentInRange(argumentValue, new Range<T>(minimum, maximum),
111     ↪ argumentName);
112
113 [Conditional("DEBUG")]
114 public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
115     ↪ Range<T> range) => Ensure.Always.ArgumentInRange(argument, range, null);
116
117 [Conditional("DEBUG")]
118 public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
119     ↪ Range<T> range, string argumentName) => Ensure.Always.ArgumentInRange(argument,
120     ↪ range, argumentName);
121
122 [Conditional("DEBUG")]
123 public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
124     ↪ Range<T> range, string argumentName, string message) =>
125     ↪ Ensure.Always.ArgumentInRange(argument, range, argumentName, message);
126
127 [Conditional("DEBUG")]
128 public static void ArgumentInRange<T>(this EnsureOnDebugExtensionRoot root, T argument,
129     ↪ Range<T> range, string argumentName, Func<string> messageBuilder) =>
130     ↪ Ensure.Always.ArgumentInRange(argument, range, argumentName, messageBuilder);
131
132 #endregion
133 }
134 }

```

./Range.cs

```
1 using System;
2 using System.Collections.Generic;
3 using Platform.Exceptions;
4
5 namespace Platform.Ranges
6 {
7     /// <summary>
8     /// Represents a range between mininum and maximum values.
9     /// Представляет диапазон между минимальным и максимальным значениями.
10    /// </summary>
11    /// <remarks>
12    /// Based on http://stackoverflow.com/questions/5343006/is-there-a-c-sharp-type-for-representing-an-integer-range
13    /// </remarks>
14    public struct Range<T> : IEquatable<Range<T>>
15    {
16        private static readonly Comparer<T> _comparer = Comparer<T>.Default;
17        private static readonly EqualityComparer<T> _equalityComparer =
18            EqualityComparer<T>.Default;
19
20        /// <summary>
21        /// Returns minimum value of the range.
22        /// Возвращает минимальное значение диапазона.
23        /// </summary>
24        public readonly T Minimum;
25
26        /// <summary>
27        /// Returns maximum value of the range.
28        /// Возвращает максимальное значение диапазона.
29        /// </summary>
30        public readonly T Maximum;
31
32        /// <summary>
33        /// Initializes a new instance of the Range class.
34        /// Инициализирует новый экземпляр класса Range.
35        /// </summary>
36        /// <param name="minimumAndMaximum">Single value for both Minimum and Maximum fields.
37        /// Одно значение для полей Minimum и Maximum.</param>
38        public Range(T minimumAndMaximum)
39        {
40            Minimum = minimumAndMaximum;
41            Maximum = minimumAndMaximum;
42        }
43
44        /// <summary>
45        /// Initializes a new instance of the Range class.
46        /// Инициализирует новый экземпляр класса Range.
47        /// </summary>
48        /// <param name="minimum">The minimum value of the range. Минимальное значение
49        /// диапазона.</param>
50        /// <param name="maximum">The maximum value of the range. Максимальное значение
51        /// диапазона.</param>
52        /// <exception cref="ArgumentException">Thrown when maximum is less than
53        /// minimum.</exception>
54        public Range(T minimum, T maximum)
55        {
56            Ensure.Always.MaximumArgumentIsGreaterOrEqualToMinimum(minimum, maximum,
57                nameof(maximum));
58            Minimum = minimum;
59            Maximum = maximum;
60        }
61
62        /// <summary>
63        /// Presents the Range in readable format.
64        /// Представляет диапазон в удобном для чтения формате.
65        /// </summary>
66        /// <returns>String representation of the Range. Строковое представление
67        /// диапазона.</returns>
68        public override string ToString() => $"{Minimum}, {Maximum}";
69
70        /// <summary>
71        /// Determines if the provided value is inside the range.
72        /// Определяет, находится ли указанное значение внутри диапазона.
73        /// </summary>
74        /// <param name="value">The value to test. Значение для проверки.</param>
75        /// <returns>True if the value is inside Range, else false. True, если значение
76        /// находится внутри диапазона, иначе false.</returns>
77    }
78 }
```

```

69 public bool ContainsValue(T value) => _comparer.Compare(Minimum, value) <= 0 &&
    ↳ _comparer.Compare(Maximum, value) >= 0;
70
71 /// <summary>
72 /// Determines if this Range is inside the bounds of another range.
73 /// Определяет, находится ли этот диапазон в пределах другого диапазона.
74 /// </summary>
75 /// <param name="range">The parent range to test on. Родительский диапазон для
    ↳ проверки.</param>
76 /// <returns>True if range is inclusive, else false. True, если диапазон включен, иначе
    ↳ false.</returns>
77 public bool IsInsideRange(Range<T> range) => range.ContainsRange(this);
78
79 /// <summary>
80 /// Determines if another range is inside the bounds of this range.
81 /// Определяет, находится ли другой диапазон внутри границ этого диапазона.
82 /// </summary>
83 /// <param name="range">The child range to test. Дочерний диапазон для проверки.</param>
84 /// <returns>True if range is inside, else false. True, если диапазон находится внутри,
    ↳ иначе false.</returns>
85 public bool ContainsRange(Range<T> range) => ContainsValue(range.Minimum) &&
    ↳ ContainsValue(range.Maximum);
86
87 /// <summary>
88 /// Indicates whether the current range is equal to another range.
89 /// Определяет, равен ли текущий диапазон другому диапазону.
90 /// </summary>
91 /// <param name="other">A range to compare with this range. Диапазон для сравнения с
    ↳ этим диапазоном.</param>
92 /// <returns>True if the current range is equal to the other range; otherwise, false.
    ↳ True, если текущий диапазон равен другому диапазону; иначе false.</returns>
93 public bool Equals(Range<T> other) => _equalityComparer.Equals(Minimum, other.Minimum)
    ↳ && _equalityComparer.Equals(Maximum, other.Maximum);
94 }
95 }

```

Index

- ./EnsureExtensions.cs, 1
- ./Range.cs, 3