

LinksPlatform's Platform.Data Class Library

1.1 ./csharp/Platform.Data/Exceptions/ArgumentLinkDoesNotExistsException.cs

```
1  using System;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Exceptions
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the argument link does not exists exception.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="ArgumentException"/>
15     public class ArgumentLinkDoesNotExistsException<TLinkAddress> : ArgumentException
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="link">
24         /// <para>A link.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="argumentName">
28         /// <para>A argument name.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public ArgumentLinkDoesNotExistsException(TLinkAddress link, string argumentName) :
33             base(FormatMessage(link, argumentName), argumentName) { }
34
35         /// <summary>
36         /// <para>
37         /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException"/> instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="link">
42         /// <para>A link.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public ArgumentLinkDoesNotExistsException(TLinkAddress link) : base(FormatMessage(link))
47             => { }
48
49         /// <summary>
50         /// <para>
51         /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException"/> instance.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         /// <param name="message">
56         /// <para>A message.</para>
57         /// <para></para>
58         /// </param>
59         /// <param name="innerException">
60         /// <para>A inner exception.</para>
61         /// <para></para>
62         /// </param>
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         public ArgumentLinkDoesNotExistsException(string message, Exception innerException) :
65             base(message, innerException) { }
66
67         /// <summary>
68         /// <para>
69         /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException"/> instance.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         /// <param name="message">
74         /// <para>A message.</para>
75         /// <para></para>
76         /// </param>
```

```

74 [MethodImpl(MethodImplOptions.AggressiveInlining)]
75 public ArgumentLinkDoesNotExistsException(string message) : base(message) { }
76
77 /// <summary>
78 /// <para>
79 /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException"/> instance.
80 /// </para>
81 /// <para></para>
82 /// </summary>
83 [MethodImpl(MethodImplOptions.AggressiveInlining)]
84 public ArgumentLinkDoesNotExistsException() { }
85 [MethodImpl(MethodImplOptions.AggressiveInlining)]
86 private static string FormatMessage(TLinkAddress link, string argumentName) => $"Связь
87   ↳ [{link}] переданная в аргумент [{argumentName}] не существует.";
88 [MethodImpl(MethodImplOptions.AggressiveInlining)]
89 private static string FormatMessage(TLinkAddress link) => $"Связь [{link}] переданная в
90   ↳ качестве аргумента не существует.";
91 }
92 }

```

1.2 ./csharp/Platform.Data/Exceptions/ArgumentLinkHasDependenciesException.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Exceptions
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the argument link has dependencies exception.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="ArgumentException"/>
15    public class ArgumentLinkHasDependenciesException<TLinkAddress> : ArgumentException
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="link">
24        /// <para>A link.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="paramName">
28        /// <para>A param name.</para>
29        /// <para></para>
30        /// </param>
31        [MethodImpl(MethodImplOptions.AggressiveInlining)]
32        public ArgumentLinkHasDependenciesException(TLinkAddress link, string paramName) :
33            ↳ base(FormatMessage(link, paramName), paramName) { }
34
35        /// <summary>
36        /// <para>
37        /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
38        /// </para>
39        /// <para></para>
40        /// </summary>
41        /// <param name="link">
42        /// <para>A link.</para>
43        /// <para></para>
44        /// </param>
45        [MethodImpl(MethodImplOptions.AggressiveInlining)]
46        public ArgumentLinkHasDependenciesException(TLinkAddress link) :
47            ↳ base(FormatMessage(link)) { }
48
49        /// <summary>
50        /// <para>
51        /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
52        /// </para>
53        /// <para></para>
54        /// </summary>
55        /// <param name="message">
56        /// <para>A message.</para>
57        /// <para></para>
58        /// </param>

```

```

56     /// </param>
57     /// <param name="innerException">
58     /// <para>A inner exception.</para>
59     /// <para></para>
60     /// </param>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public ArgumentLinkHasDependenciesException(string message, Exception innerException) :
        ↳ base(message, innerException) { }
63
64     /// <summary>
65     /// <para>
66     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="message">
71     /// <para>A message.</para>
72     /// <para></para>
73     /// </param>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public ArgumentLinkHasDependenciesException(string message) : base(message) { }
76
77     /// <summary>
78     /// <para>
79     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     public ArgumentLinkHasDependenciesException() { }
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     private static string FormatMessage(TLinkAddress link, string paramName) => $"У связи
        ↳ [{link}] переданной в аргумент [{paramName}] присутствуют зависимости, которые
        ↳ препятствуют изменению её внутренней структуры.";
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     private static string FormatMessage(TLinkAddress link) => $"У связи [{link}] переданной
        ↳ в качестве аргумента присутствуют зависимости, которые препятствуют изменению её
        ↳ внутренней структуры.";
89 }
90 }

```

1.3 ./csharp/Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsException.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Exceptions
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the link with same value already exists exception.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="Exception"/>
15     public class LinkWithSameValueAlreadyExistsException : Exception
16     {
17         /// <summary>
18         /// <para>
19         /// The default message.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         public static readonly string DefaultMessage = "Связь с таким же значением уже
            ↳ существует.";
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="message">
32         /// <para>A message.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="innerException">

```

```

36     /// <para>A inner exception.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public LinkWithSameValueAlreadyExistsException(string message, Exception innerException)
41         ↪ : base(message, innerException) { }
42
43     /// <summary>
44     /// <para>
45     /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="message">
50     /// <para>A message.</para>
51     /// <para></para>
52     /// </param>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public LinkWithSameValueAlreadyExistsException(string message) : base(message) { }
55
56     /// <summary>
57     /// <para>
58     /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public LinkWithSameValueAlreadyExistsException() : base(DefaultMessage) { }
64 }

```

1.4 ./csharp/Platform.Data/Exceptions/LinksLimitReachedException.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Exceptions
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links limit reached exception.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksLimitReachedExceptionBase"/>
15     public class LinksLimitReachedException<TLinkAddress> : LinksLimitReachedExceptionBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksLimitReachedException"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="limit">
24         /// <para>A limit.</para>
25         /// <para></para>
26         /// </param>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public LinksLimitReachedException(TLinkAddress limit) : this(FormatMessage(limit)) { }
29
30         /// <summary>
31         /// <para>
32         /// Initializes a new <see cref="LinksLimitReachedException"/> instance.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="message">
37         /// <para>A message.</para>
38         /// <para></para>
39         /// </param>
40         /// <param name="innerException">
41         /// <para>A inner exception.</para>
42         /// <para></para>
43         /// </param>
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public LinksLimitReachedException(string message, Exception innerException) :
46             ↪ base(message, innerException) { }

```

```

46
47     /// <summary>
48     /// <para>
49     /// Initializes a new <see cref="LinksLimitReachedException"/> instance.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="message">
54     /// <para>A message.</para>
55     /// <para></para>
56     /// </param>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public LinksLimitReachedException(string message) : base(message) { }
59
60     /// <summary>
61     /// <para>
62     /// Initializes a new <see cref="LinksLimitReachedException"/> instance.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public LinksLimitReachedException() : base(DefaultMessage) { }
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     private static string FormatMessage(TLinkAddress limit) => $"Достигнут лимит количества
    ↳ связей в хранилище ({limit}).";
70 }
71 }

```

1.5 ./csharp/Platform.Data/Exceptions/LinksLimitReachedExceptionBase.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Exceptions
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the links limit reached exception base.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="Exception"/>
15    public abstract class LinksLimitReachedExceptionBase : Exception
16    {
17        /// <summary>
18        /// <para>
19        /// The default message.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        public static readonly string DefaultMessage = "Достигнут лимит количества связей в
    ↳ хранилище.";
24
25        /// <summary>
26        /// <para>
27        /// Initializes a new <see cref="LinksLimitReachedExceptionBase"/> instance.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        /// <param name="message">
32        /// <para>A message.</para>
33        /// <para></para>
34        /// </param>
35        /// <param name="innerException">
36        /// <para>A inner exception.</para>
37        /// <para></para>
38        /// </param>
39        [MethodImpl(MethodImplOptions.AggressiveInlining)]
40        protected LinksLimitReachedExceptionBase(string message, Exception innerException) :
    ↳ base(message, innerException) { }
41
42        /// <summary>
43        /// <para>
44        /// Initializes a new <see cref="LinksLimitReachedExceptionBase"/> instance.
45        /// </para>
46        /// <para></para>
47        /// </summary>

```

```

48     /// <param name="message">
49     /// <para>A message.</para>
50     /// <para></para>
51     /// </param>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected LinksLimitReachedExceptionBase(string message) : base(message) { }
54 }
55 }

```

1.6 ./csharp/Platform.Data/Hybrid.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Exceptions;
5  using Platform.Reflection;
6  using Platform.Converters;
7  using Platform.Numbers;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data
12 {
13     /// <summary>
14     /// <para>
15     /// The hybrid.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public struct Hybrid<TLinkAddress> : IEquatable<Hybrid<TLinkAddress>>
20     {
21         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
22             EqualityComparer<TLinkAddress>.Default;
23         private static readonly UncheckedSignExtendingConverter<TLinkAddress, long>
24             _addressToInt64Converter = UncheckedSignExtendingConverter<TLinkAddress,
25             long>.Default;
26         private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
27             = UncheckedConverter<long, TLinkAddress>.Default;
28         private static readonly UncheckedConverter<TLinkAddress, ulong>
29             _addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
30         private static readonly UncheckedConverter<ulong, TLinkAddress>
31             _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
32         private static readonly UncheckedConverter<object, long> _objectToInt64Converter =
33             UncheckedConverter<object, long>.Default;
34
35         /// <summary>
36         /// <para>
37         /// The max value.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         public static readonly ulong HalfOfNumberValuesRange =
42             _addressToUInt64Converter.Convert(NumericType<TLinkAddress>.MaxValue) / 2;
43         /// <summary>
44         /// <para>
45         /// The half of number values range.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public static readonly TLinkAddress ExternalZero =
50             _uInt64ToAddressConverter.Convert(HalfOfNumberValuesRange + 1UL);
51
52         /// <summary>
53         /// <para>
54         /// The value.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         public readonly TLinkAddress Value;
59
60         /// <summary>
61         /// <para>
62         /// Gets the is nothing value.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         public bool IsNothing
67         {
68             [MethodImpl(MethodImplOptions.AggressiveInlining)]
69             get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue == 0;
70         }
71     }
72 }

```

```

62
63     /// <summary>
64     /// <para>
65     /// Gets the is internal value.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public bool IsInternal
70     {
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         get => SignedValue > 0;
73     }
74
75     /// <summary>
76     /// <para>
77     /// Gets the is external value.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     public bool IsExternal
82     {
83         [MethodImpl(MethodImplOptions.AggressiveInlining)]
84         get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue < 0;
85     }
86
87     /// <summary>
88     /// <para>
89     /// Gets the signed value value.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     public long SignedValue
94     {
95         [MethodImpl(MethodImplOptions.AggressiveInlining)]
96         get => _addressToInt64Converter.Convert(Value);
97     }
98
99     /// <summary>
100    /// <para>
101    /// Gets the absolute value value.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    public long AbsoluteValue
106    {
107        [MethodImpl(MethodImplOptions.AggressiveInlining)]
108        get => _equalityComparer.Equals(Value, ExternalZero) ? 0 :
            ↪ Platform.Numbers.Math.Abs(SignedValue);
109    }
110
111    /// <summary>
112    /// <para>
113    /// Initializes a new <see cref="Hybrid"/> instance.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="value">
118    /// <para>A value.</para>
119    /// <para></para>
120    /// </param>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    public Hybrid(TLinkAddress value)
123    {
124        Ensure.OnDebug.IsUnsignedInteger<TLinkAddress>();
125        Value = value;
126    }
127
128    /// <summary>
129    /// <para>
130    /// Initializes a new <see cref="Hybrid"/> instance.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="value">
135    /// <para>A value.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="isExternal">

```

```

139 /// <para>A is external.</para>
140 /// <para></para>
141 /// </param>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 public Hybrid(TLinkAddress value, bool isExternal)
144 {
145     if (_equalityComparer.Equals(value, default) && isExternal)
146     {
147         Value = ExternalZero;
148     }
149     else
150     {
151         if (isExternal)
152         {
153             Value = Math<TLinkAddress>.Negate(value);
154         }
155         else
156         {
157             Value = value;
158         }
159     }
160 }
161
162 /// <summary>
163 /// <para>
164 /// Initializes a new <see cref="Hybrid"/> instance.
165 /// </para>
166 /// <para></para>
167 /// </summary>
168 /// <param name="value">
169 /// <para>A value.</para>
170 /// <para></para>
171 /// </param>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 public Hybrid(object value) => Value =
174     ↪ _int64ToAddressConverter.Convert(_objectToInt64Converter.Convert(value));
175
176 /// <summary>
177 /// <para>
178 /// Initializes a new <see cref="Hybrid"/> instance.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 /// <param name="value">
183 /// <para>A value.</para>
184 /// <para></para>
185 /// </param>
186 /// <param name="isExternal">
187 /// <para>A is external.</para>
188 /// <para></para>
189 /// </param>
190 [MethodImpl(MethodImplOptions.AggressiveInlining)]
191 public Hybrid(object value, bool isExternal)
192 {
193     var signedValue = value == null ? 0 : _objectToInt64Converter.Convert(value);
194     if (signedValue == 0 && isExternal)
195     {
196         Value = ExternalZero;
197     }
198     else
199     {
200         var absoluteValue = System.Math.Abs(signedValue);
201         Value = isExternal ? _int64ToAddressConverter.Convert(-absoluteValue) :
202             ↪ _int64ToAddressConverter.Convert(absoluteValue);
203     }
204 }
205
206 [MethodImpl(MethodImplOptions.AggressiveInlining)]
207 public static implicit operator Hybrid<TLinkAddress>(TLinkAddress integer) => new
208     ↪ Hybrid<TLinkAddress>(integer);
209
210 [MethodImpl(MethodImplOptions.AggressiveInlining)]
211 public static explicit operator Hybrid<TLinkAddress>(ulong integer) => new
212     ↪ Hybrid<TLinkAddress>(integer);
213
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 public static explicit operator Hybrid<TLinkAddress>(long integer) => new
216     ↪ Hybrid<TLinkAddress>(integer);

```



```

212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public static explicit operator Hybrid<TLinkAddress>(uint integer) => new
214     ↳ Hybrid<TLinkAddress>(integer);
215
216 [MethodImpl(MethodImplOptions.AggressiveInlining)]
217 public static explicit operator Hybrid<TLinkAddress>(int integer) => new
218     ↳ Hybrid<TLinkAddress>(integer);
219
220 [MethodImpl(MethodImplOptions.AggressiveInlining)]
221 public static explicit operator Hybrid<TLinkAddress>(ushort integer) => new
222     ↳ Hybrid<TLinkAddress>(integer);
223
224 [MethodImpl(MethodImplOptions.AggressiveInlining)]
225 public static explicit operator Hybrid<TLinkAddress>(short integer) => new
226     ↳ Hybrid<TLinkAddress>(integer);
227
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 public static explicit operator Hybrid<TLinkAddress>(sbyte integer) => new
230     ↳ Hybrid<TLinkAddress>(integer);
231
232 [MethodImpl(MethodImplOptions.AggressiveInlining)]
233 public static implicit operator TLinkAddress(Hybrid<TLinkAddress> hybrid) =>
234     ↳ hybrid.Value;
235
236 [MethodImpl(MethodImplOptions.AggressiveInlining)]
237 public static explicit operator ulong(Hybrid<TLinkAddress> hybrid) =>
238     ↳ CheckedConverter<TLinkAddress, ulong>.Default.Convert(hybrid.Value);
239
240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
241 public static explicit operator long(Hybrid<TLinkAddress> hybrid) =>
242     ↳ hybrid.AbsoluteValue;
243
244 [MethodImpl(MethodImplOptions.AggressiveInlining)]
245 public static explicit operator uint(Hybrid<TLinkAddress> hybrid) =>
246     ↳ CheckedConverter<TLinkAddress, uint>.Default.Convert(hybrid.Value);
247
248 [MethodImpl(MethodImplOptions.AggressiveInlining)]
249 public static explicit operator int(Hybrid<TLinkAddress> hybrid) =>
250     ↳ (int)hybrid.AbsoluteValue;
251
252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
253 public static explicit operator ushort(Hybrid<TLinkAddress> hybrid) =>
254     ↳ CheckedConverter<TLinkAddress, ushort>.Default.Convert(hybrid.Value);
255
256 [MethodImpl(MethodImplOptions.AggressiveInlining)]
257 public static explicit operator short(Hybrid<TLinkAddress> hybrid) =>
258     ↳ (short)hybrid.AbsoluteValue;
259
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 public static explicit operator byte(Hybrid<TLinkAddress> hybrid) =>
262     ↳ CheckedConverter<TLinkAddress, byte>.Default.Convert(hybrid.Value);
263
264 [MethodImpl(MethodImplOptions.AggressiveInlining)]
265 public static explicit operator sbyte(Hybrid<TLinkAddress> hybrid) =>
266     ↳ (sbyte)hybrid.AbsoluteValue;
267
268 /// <summary>
269 /// <para>
270 /// Returns the string.
271 /// </para>
272 /// <para></para>
273 /// </summary>
274 /// <returns>
275 /// <para>The string</para>
276 /// <para></para>
277 /// </returns>
278 [MethodImpl(MethodImplOptions.AggressiveInlining)]
279 public override string ToString() => IsExternal ? $"{<AbsolutePath>}" :
280     ↳ Value.ToString();
281
282 /// <summary>
283 /// <para>
284 /// Determines whether this instance equals.

```

```

274     /// </para>
275     /// <para></para>
276     /// </summary>
277     /// <param name="other">
278     /// <para>The other.</para>
279     /// <para></para>
280     /// </param>
281     /// <returns>
282     /// <para>The bool</para>
283     /// <para></para>
284     /// </returns>
285     [MethodImpl(MethodImplOptions.AggressiveInlining)]
286     public bool Equals(Hybrid<TLinkAddress> other) => _equalityComparer.Equals(Value,
        ↪ other.Value);

287
288     /// <summary>
289     /// <para>
290     /// Determines whether this instance equals.
291     /// </para>
292     /// <para></para>
293     /// </summary>
294     /// <param name="obj">
295     /// <para>The obj.</para>
296     /// <para></para>
297     /// </param>
298     /// <returns>
299     /// <para>The bool</para>
300     /// <para></para>
301     /// </returns>
302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
303     public override bool Equals(object obj) => obj is Hybrid<TLinkAddress> hybrid ?
        ↪ Equals(hybrid) : false;

304
305     /// <summary>
306     /// <para>
307     /// Gets the hash code.
308     /// </para>
309     /// <para></para>
310     /// </summary>
311     /// <returns>
312     /// <para>The int</para>
313     /// <para></para>
314     /// </returns>
315     [MethodImpl(MethodImplOptions.AggressiveInlining)]
316     public override int GetHashCode() => Value.GetHashCode();
317
318     [MethodImpl(MethodImplOptions.AggressiveInlining)]
319     public static bool operator ==(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
        ↪ left.Equals(right);

320
321     [MethodImpl(MethodImplOptions.AggressiveInlining)]
322     public static bool operator !=(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
        ↪ !(left == right);
323 }
324 }

```

1.7 ./csharp/Platform.Data/ILinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data
8  {
9      /// <summary>
10     /// <para>Represents an interface for manipulating data in the Links (links storage)
11     ↪ format.</para>
12     /// <para>Представляет интерфейс для манипуляции с данными в формате Links (хранилища
13     ↪ связей).</para>
14     /// </summary>
15     /// <remarks>
16     /// <para>This interface is independent of the size of the content of the link, meaning it
17     ↪ is suitable for both doublets, triplets, and link sequences of any size.</para>
18     /// <para>Этот интерфейс не зависит от размера содержимого связи, а значит подходит как для
19     ↪ дуплетов, триплетов и последовательностей связей любого размера.</para>
20     /// </remarks>
21     public interface ILinks<TLinkAddress, TConstants>

```

```

18     where TConstants : LinksConstants<TLinkAddress>
19 {
20     #region Constants
21
22     /// <summary>
23     /// <para>Returns the set of constants that is necessary for effective communication
24     → with the methods of this interface.</para>
25     /// <para>Возвращает набор констант, который необходим для эффективной коммуникации с
26     → методами этого интерфейса.</para>
27     /// </summary>
28     /// <remarks>
29     /// <para>These constants are not changed since the creation of the links storage access
30     → point.</para>
31     /// <para>Эти константы не меняются с момента создания точки доступа к хранилищу
32     → связей.</para>
33     /// </remarks>
34     TConstants Constants
35     {
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         get;
38     }
39
40     #endregion
41
42     #region Read
43
44     /// <summary>
45     /// <para>Counts and returns the total number of links in the storage that meet the
46     → specified restriction.</para>
47     /// <para>Подсчитывает и возвращает общее число связей находящихся в хранилище,
48     → соответствующих указанным ограничениям.</para>
49     /// </summary>
50     /// <param name="restriction"><para>Restriction on the contents of
51     → links.</para><para>Ограничения на содержимое связей.</para></param>
52     /// <returns><para>The total number of links in the storage that meet the specified
53     → restriction.</para><para>Общее число связей находящихся в хранилище, соответствующих
54     → указанным ограничениям.</para></returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     TLinkAddress Count(IList<TLinkAddress> restriction);
57
58     /// <summary>
59     /// <para>Passes through all the links matching the pattern, invoking a handler for each
60     → matching link.</para>
61     /// <para>Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
62     → (handler) для каждой подходящей связи.</para>
63     /// </summary>
64     /// <param name="handler"><para>A handler for each matching link.</para><para>Обработчик
65     → для каждой подходящей связи.</para></param>
66     /// <param name="restriction">
67     /// <para>Restriction on the contents of links. Each constraint can have values:
68     → Constants.Null - the 0th link denoting a reference to the void, Any - the absence of
69     → a constraint, 1..∞ a specific link index.</para>
70     /// <para>Ограничения на содержимое связей. Каждое ограничение может иметь значения:
71     → Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any - отсутствие
72     → ограничения, 1..∞ конкретный индекс связи.</para>
73     /// </param>
74     /// <returns><para>Constants.Continue, if the pass through the links was not
75     → interrupted, and Constants.Break otherwise.</para><para>Constants.Continue, в случае
76     → если проход по связям не был прерван и Constants.Break в обратном
77     → случае.</para></returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler, IList<TLinkAddress>
80     → restriction);
81
82     #endregion
83
84     #region Write
85
86     /// <summary>
87     /// <para>Creates a link.</para>
88     /// <para>Создаёт связь.</para>
89     /// <param name="substitution">
90     /// <para>The content of a new link. This argument is optional, if the null passed as
91     → value that means no content of a link is set.</para>
92     /// <para>Содержимое новой связи. Этот аргумент опционален, если null передан в качестве
93     → значения это означает, что никакого содержимого для связи не установлено.</para>
94     /// </param>

```

```

73     /// </summary>
74     /// <returns><para>Index of the created link.</para><para>Индекс созданной
    → связи.</para></returns>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     TLinkAddress Create(IList<TLinkAddress> substitution); // TODO: Возвращать связь
    → возвращать нужно целиком.
77
78     /// <summary>
79     /// Обновляет связь с указанными restriction[Constants.IndexPart] в адресом связи
80     /// на связь с указанным новым содержимым.
81     /// </summary>
82     /// <param name="restriction">
83     /// Ограничения на содержимое связей.
84     /// Предполагается, что будет указан индекс связи (в restriction[Constants.IndexPart]) и
    → далее за ним будет следовать содержимое связи.
85     /// Каждое ограничение может иметь значения: Constants.Null - 0-я связь, обозначающая
    → ссылку на пустоту,
86     /// Constants.Itself - требование установить ссылку на себя, 1.. $\infty$  конкретный индекс
    → другой связи.
87     /// </param>
88     /// <param name="substitution"></param>
89     /// <returns>Индекс обновлённой связи.</returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     TLinkAddress Update(IList<TLinkAddress> restriction, IList<TLinkAddress> substitution);
    → // TODO: Возможно и возвращать связь нужно целиком.
92
93     /// <summary>
94     /// <para>Deletes links that match the specified restriction.</para>
95     /// <para>Удаляет связи соответствующие указанным ограничениям.</para>
96     /// <param name="restriction">
97     /// <para>Restriction on the content of a link. This argument is optional, if the null
    → passed as value that means no restriction on the content of a link are set.</para>
98     /// <para>Ограничения на содержимое связи. Этот аргумент опционален, если null передан в
    → качестве значения это означает, что никаких ограничений на содержимое связи не
    → установлено.</para>
99     /// </param>
100    /// </summary>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    void Delete(IList<TLinkAddress> restriction); // TODO: Возможно стоит возвращать
    → удалённую связь, если удаление было реально выполнено, и Constants.Null, если нет.
103
104    #endregion
105    }
106 }

```

1.8 ./csharp/Platform.Data/ILinksExtensions.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Setters;
5  using Platform.Data.Exceptions;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the links extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class ILinksExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// Counts the links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <typeparam name="TLinkAddress">
26         /// <para>The link address.</para>
27         /// <para></para>
28         /// </typeparam>
29         /// <typeparam name="TConstants">
30         /// <para>The constants.</para>
31         /// <para></para>
32         /// </typeparam>
33         /// <param name="links">

```

```

34    /// <para>The links.</para>
35    /// <para></para>
36    /// </param>
37    /// <param name="restrictions">
38    /// <para>The restrictions.</para>
39    /// <para></para>
40    /// </param>
41    /// <returns>
42    /// <para>The link address</para>
43    /// <para></para>
44    /// </returns>
45    [MethodImpl(MethodImplOptions.AggressiveInlining)]
46    public static TLinkAddress Count<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    → TConstants> links, params TLinkAddress[] restrictions)
47        where TConstants : LinksConstants<TLinkAddress>
48        => links.Count(restrictions);
49
50    /// <summary>
51    /// Возвращает значение, определяющее существует ли связь с указанным индексом в
    → хранилище связей.
52    /// </summary>
53    /// <param name="links">Хранилище связей.</param>
54    /// <param name="link">Индекс проверяемой на существование связи.</param>
55    /// <returns>Значение, определяющее существует ли связь.</returns>
56    [MethodImpl(MethodImplOptions.AggressiveInlining)]
57    public static bool Exists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    → TConstants> links, TLinkAddress link)
58        where TConstants : LinksConstants<TLinkAddress>
59    {
60        var constants = links.Constants;
61        return constants.IsExternalReference(link) || (constants.IsInternalReference(link)
    → && Comparer<TLinkAddress>.Default.Compare(links.Count(new
    → LinkAddress<TLinkAddress>(link)), default) > 0);
62    }
63
64    /// <param name="links">Хранилище связей.</param>
65    /// <param name="link">Индекс проверяемой на существование связи.</param>
66    /// <remarks>
67    /// TODO: May be move to EnsureExtensions or make it both there and here
68    /// </remarks>
69    [MethodImpl(MethodImplOptions.AggressiveInlining)]
70    public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    → TConstants> links, TLinkAddress link)
71        where TConstants : LinksConstants<TLinkAddress>
72    {
73        if (!links.Exists(link))
74        {
75            throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link);
76        }
77    }
78
79    /// <param name="links">Хранилище связей.</param>
80    /// <param name="link">Индекс проверяемой на существование связи.</param>
81    /// <param name="argumentName">Имя аргумента, в который передаётся индекс связи.</param>
82    [MethodImpl(MethodImplOptions.AggressiveInlining)]
83    public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    → TConstants> links, TLinkAddress link, string argumentName)
84        where TConstants : LinksConstants<TLinkAddress>
85    {
86        if (!links.Exists(link))
87        {
88            throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
89        }
90    }
91
92    /// <summary>
93    /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    → (handler) для каждой подходящей связи.
94    /// </summary>
95    /// <param name="links">Хранилище связей.</param>
96    /// <param name="handler">Обработчик каждой подходящей связи.</param>
97    /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
    → может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
    → Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
98    /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    → случае.</returns>
99    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

public static TLinkAddress Each<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, Func<IList<TLinkAddress>, TLinkAddress> handler, params
    ↳ TLinkAddress[] restrictions)
    where TConstants : LinksConstants<TLinkAddress>
    => links.Each(handler, restrictions);

/// <summary>
/// Возвращает части-значения для связи с указанным индексом.
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="link">Индекс связи.</param>
/// <returns>Уникальную связь.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IList<TLinkAddress> GetLink<TLinkAddress, TConstants>(this
    ↳ ILinks<TLinkAddress, TConstants> links, TLinkAddress link)
    where TConstants : LinksConstants<TLinkAddress>
{
    var constants = links.Constants;
    if (constants.IsExternalReference(link))
    {
        return new Point<TLinkAddress>(link, constants.TargetPart + 1);
    }
    var linkPartsSetter = new Setter<IList<TLinkAddress>,
        ↳ TLinkAddress>(constants.Continue, constants.Break);
    links.Each(linkPartsSetter.SetAndReturnTrue, link);
    return linkPartsSetter.Result;
}

#region Points

/// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
    ↳ точкой полностью (связью замкнутой на себе дважды).</summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="link">Индекс проверяемой связи.</param>
/// <returns>Значение, определяющее является ли связь точкой полностью.</returns>
/// <remarks>
/// Связь точка - это связь, у которой начало (Source) и конец (Target) есть сама эта
    ↳ связь.
/// Но что, если точка уже есть, а нужно создать пару с таким же значением? Должны ли
    ↳ точка и пара существовать одновременно?
/// Или в качестве решения для точек нужно использовать 0 в качестве начала и конца, а
    ↳ сортировать по индексу в массиве связей?
/// Какое тогда будет значение Source и Target у точки? 0 или её индекс?
/// Или точка должна быть одновременно точкой и парой, а также последовательностями из
    ↳ самой себя любого размера?
/// Как только есть ссылка на себя, появляется этот парадокс, причём достаточно даже
    ↳ одной ссылки на себя (частичной точки).
/// А что если не выбирать что является точкой, пара нулей (цикл через пустоту) или
    ↳ самостоятельный цикл через себя? Что если предоставить все варианты использования
    ↳ связей?
/// Что если разрешить и нули, а так же частичные варианты?
///
/// Что если точка, это только в том случае когда link.Source == link &&
    ↳ link.Target == link , т.е. дважды ссылка на себя.
/// А пара это тогда, когда link.Source == link.Target && link.Source != link ,
    ↳ т.е. ссылка не на себя а во вне.
///
/// Тогда если у нас уже создана пара, но нам нужна точка, мы можем используя
    ↳ промежуточную связь,
/// например "DoubletOf" обозначить что является точно парой, а что точно точкой.
/// И наоборот этот же метод поможет, если уже существует точка, но нам нужна пара.
/// </remarks>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool IsFullPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, TLinkAddress link)
    where TConstants : LinksConstants<TLinkAddress>
{
    if (links.Constants.IsExternalReference(link))
    {
        return true;
    }
    links.EnsureLinkExists(link);
    return Point<TLinkAddress>.IsFullPoint(links.GetLink(link));
}

/// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
    ↳ точкой частично (связью замкнутой на себе как минимум один раз).</summary>
/// <param name="links">Хранилище связей.</param>

```

```

162     /// <param name="link">Индекс проверяемой связи.</param>
163     /// <returns>Значение, определяющее является ли связь точкой частично.</returns>
164     /// <remarks>
165     /// Достаточно любой одной ссылки на себя.
166     /// Также в будущем можно будет проверять и всех родителей, чтобы проверить есть ли
    ↪ ссылки на себя (на эту связь).
167     /// </remarks>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     public static bool IsPartialPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↪ TConstants> links, TLinkAddress link)
170     where TConstants : LinksConstants<TLinkAddress>
171     {
172         if (links.Constants.IsExternalReference(link))
173         {
174             return true;
175         }
176         links.EnsureLinkExists(link);
177         return Point<TLinkAddress>.IsPartialPoint(links.GetLink(link));
178     }
179
180     #endregion
181 }
182 }

```

1.9 ./csharp/Platform.Data/ISynchronizedLinks.cs

```

1 using Platform.Threading.Synchronization;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data
6 {
7     /// <summary>
8     /// <para>
9     /// Defines the synchronized links.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="ISynchronized{TLinks}">
14    /// <seealso cref="ILinks{TLinkAddress, TConstants}">
15    public interface ISynchronizedLinks<TLinkAddress, TLinks, TConstants> :
    ↪ ISynchronized<TLinks>, ILinks<TLinkAddress, TConstants>
16    where TLinks : ILinks<TLinkAddress, TConstants>
17    where TConstants : LinksConstants<TLinkAddress>
18    {
19    }
20 }

```

1.10 ./csharp/Platform.Data/LinkAddress.cs

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Runtime.CompilerServices;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data
9 {
10    /// <summary>
11    /// <para>
12    /// Represents the link address.
13    /// </para>
14    /// <para></para>
15    /// </summary>
16    /// <seealso cref="IEquatable{LinkAddress{TLinkAddress}}">
17    /// <seealso cref="IList{TLinkAddress}">
18    public class LinkAddress<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>,
    ↪ IList<TLinkAddress>
19    {
20        private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
    ↪ EqualityComparer<TLinkAddress>.Default;
21
22        /// <summary>
23        /// <para>
24        /// Gets the index value.
25        /// </para>
26        /// <para></para>
27        /// </summary>
28        public TLinkAddress Index
29        {

```

```

30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         get;
32     }
33
34     /// <summary>
35     /// <para>
36     /// The not supported exception.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     public TLinkAddress this[int index]
41     {
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         get
44         {
45             if (index == 0)
46             {
47                 return Index;
48             }
49             else
50             {
51                 throw new IndexOutOfRangeException();
52             }
53         }
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         set => throw new NotSupportedException();
56     }
57
58     /// <summary>
59     /// <para>
60     /// Gets the count value.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     public int Count
65     {
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         get => 1;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Gets the is read only value.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public bool IsReadOnly
77     {
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         get => true;
80     }
81
82     /// <summary>
83     /// <para>
84     /// Initializes a new <see cref="LinkAddress"/> instance.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="index">
89     /// <para>A index.</para>
90     /// <para></para>
91     /// </param>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public LinkAddress(TLinkAddress index) => Index = index;
94
95     /// <summary>
96     /// <para>
97     /// Adds the item.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="item">
102    /// <para>The item.</para>
103    /// <para></para>
104    /// </param>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    public void Add(TLinkAddress item) => throw new NotSupportedException();
107
108    /// <summary>

```



```

109     /// <para>
110     /// Clears this instance.
111     /// </para>
112     /// <para></para>
113     /// </summary>
114     [MethodImpl(MethodImplOptions.AggressiveInlining)]
115     public void Clear() => throw new NotSupportedException();
116
117     /// <summary>
118     /// <para>
119     /// Determines whether this instance contains.
120     /// </para>
121     /// <para></para>
122     /// </summary>
123     /// <param name="item">
124     /// <para>The item.</para>
125     /// <para></para>
126     /// </param>
127     /// <returns>
128     /// <para>The bool</para>
129     /// <para></para>
130     /// </returns>
131     [MethodImpl(MethodImplOptions.AggressiveInlining)]
132     public virtual bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index)
133     ↪ ? true : false;
134
135     /// <summary>
136     /// <para>
137     /// Copies the to using the specified array.
138     /// </para>
139     /// <para></para>
140     /// </summary>
141     /// <param name="array">
142     /// <para>The array.</para>
143     /// <para></para>
144     /// </param>
145     /// <param name="arrayIndex">
146     /// <para>The array index.</para>
147     /// <para></para>
148     /// </param>
149     [MethodImpl(MethodImplOptions.AggressiveInlining)]
150     public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
151
152     /// <summary>
153     /// <para>
154     /// Gets the enumerator.
155     /// </para>
156     /// <para></para>
157     /// </summary>
158     /// <returns>
159     /// <para>An enumerator of t link address</para>
160     /// <para></para>
161     /// </returns>
162     [MethodImpl(MethodImplOptions.AggressiveInlining)]
163     public IEnumerator<TLinkAddress> GetEnumerator()
164     {
165         yield return Index;
166     }
167
168     /// <summary>
169     /// <para>
170     /// Indexes the of using the specified item.
171     /// </para>
172     /// <para></para>
173     /// </summary>
174     /// <param name="item">
175     /// <para>The item.</para>
176     /// <para></para>
177     /// </param>
178     /// <returns>
179     /// <para>The int</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
184     ↪ 0 : -1;
185
186     /// <summary>

```

```

185     /// <para>
186     /// Inserts the index.
187     /// </para>
188     /// <para></para>
189     /// </summary>
190     /// <param name="index">
191     /// <para>The index.</para>
192     /// <para></para>
193     /// </param>
194     /// <param name="item">
195     /// <para>The item.</para>
196     /// <para></para>
197     /// </param>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance remove.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="item">
208     /// <para>The item.</para>
209     /// <para></para>
210     /// </param>
211     /// <returns>
212     /// <para>The bool</para>
213     /// <para></para>
214     /// </returns>
215     [MethodImpl(MethodImplOptions.AggressiveInlining)]
216     public bool Remove(TLinkAddress item) => throw new NotSupportedException();
217
218     /// <summary>
219     /// <para>
220     /// Removes the at using the specified index.
221     /// </para>
222     /// <para></para>
223     /// </summary>
224     /// <param name="index">
225     /// <para>The index.</para>
226     /// <para></para>
227     /// </param>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     public void RemoveAt(int index) => throw new NotSupportedException();
230
231     /// <summary>
232     /// <para>
233     /// Gets the enumerator.
234     /// </para>
235     /// <para></para>
236     /// </summary>
237     /// <returns>
238     /// <para>The enumerator</para>
239     /// <para></para>
240     /// </returns>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     IEnumerator IEnumerable.GetEnumerator()
243     {
244         yield return Index;
245     }
246
247     /// <summary>
248     /// <para>
249     /// Determines whether this instance equals.
250     /// </para>
251     /// <para></para>
252     /// </summary>
253     /// <param name="other">
254     /// <para>The other.</para>
255     /// <para></para>
256     /// </param>
257     /// <returns>
258     /// <para>The bool</para>
259     /// <para></para>
260     /// </returns>
261     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

262 public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
    ↳ _equalityComparer.Equals(Index, other.Index);
263
264 [MethodImpl(MethodImplOptions.AggressiveInlining)]
265 public static implicit operator TLinkAddress(LinkAddress<TLinkAddress> linkAddress) =>
    ↳ linkAddress.Index;
266
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 public static implicit operator LinkAddress<TLinkAddress>(TLinkAddress linkAddress) =>
    ↳ new LinkAddress<TLinkAddress>(linkAddress);
269
270 /// <summary>
271 /// <para>
272 /// Determines whether this instance equals.
273 /// </para>
274 /// <para></para>
275 /// </summary>
276 /// <param name="obj">
277 /// <para>The obj.</para>
278 /// <para></para>
279 /// </param>
280 /// <returns>
281 /// <para>The bool</para>
282 /// <para></para>
283 /// </returns>
284 [MethodImpl(MethodImplOptions.AggressiveInlining)]
285 public override bool Equals(object obj) => obj is LinkAddress<TLinkAddress> linkAddress
    ↳ ? Equals(linkAddress) : false;
286
287 /// <summary>
288 /// <para>
289 /// Gets the hash code.
290 /// </para>
291 /// <para></para>
292 /// </summary>
293 /// <returns>
294 /// <para>The int</para>
295 /// <para></para>
296 /// </returns>
297 [MethodImpl(MethodImplOptions.AggressiveInlining)]
298 public override int GetHashCode() => Index.GetHashCode();
299
300 /// <summary>
301 /// <para>
302 /// Returns the string.
303 /// </para>
304 /// <para></para>
305 /// </summary>
306 /// <returns>
307 /// <para>The string</para>
308 /// <para></para>
309 /// </returns>
310 [MethodImpl(MethodImplOptions.AggressiveInlining)]
311 public override string ToString() => Index.ToString();
312
313 [MethodImpl(MethodImplOptions.AggressiveInlining)]
314 public static bool operator ==(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
    ↳ right)
315 {
316     if (left == null && right == null)
317     {
318         return true;
319     }
320     if (left == null)
321     {
322         return false;
323     }
324     return left.Equals(right);
325 }
326
327 [MethodImpl(MethodImplOptions.AggressiveInlining)]
328 public static bool operator !=(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
    ↳ right) => !(left == right);
329
330 }

```

1.11 ./csharp/Platform.Data/LinksConstants.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Ranges;
3  using Platform.Reflection;
4  using Platform.Converters;
5  using Platform.Numbers;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the links constants.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksConstantsBase"/>
18     public class LinksConstants<TLinkAddress> : LinksConstantsBase
19     {
20         private static readonly TLinkAddress _one = Arithmetic<TLinkAddress>.Increment(default);
21         private static readonly UncheckedConverter<ulong, TLinkAddress>
22             ↪ _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
23
24         #region Link parts
25
26         /// <summary>Возвращает индекс части, которая отвечает за индекс (адрес, идентификатор)
27         ↪ самой связи.</summary>
28         public int IndexPart
29         {
30             [MethodImpl(MethodImplOptions.AggressiveInlining)]
31             get;
32         }
33
34         /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-начало (первая
35         ↪ часть-значение).</summary>
36         public int SourcePart
37         {
38             [MethodImpl(MethodImplOptions.AggressiveInlining)]
39             get;
40         }
41
42         /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-конец
43         ↪ (последняя часть-значение).</summary>
44         public int TargetPart
45         {
46             [MethodImpl(MethodImplOptions.AggressiveInlining)]
47             get;
48         }
49
50         #endregion
51
52         #region Flow control
53
54         /// <summary>Возвращает значение, обозначающее продолжение прохода по связям.</summary>
55         /// <remarks>Используется в функции обработчике, который передаётся в функцию
56         ↪ Each.</remarks>
57         public TLinkAddress Continue
58         {
59             [MethodImpl(MethodImplOptions.AggressiveInlining)]
60             get;
61         }
62
63         /// <summary>Возвращает значение, обозначающее пропуск в проходе по связям.</summary>
64         public TLinkAddress Skip
65         {
66             [MethodImpl(MethodImplOptions.AggressiveInlining)]
67             get;
68         }
69
70         /// <summary>Возвращает значение, обозначающее остановку прохода по связям.</summary>
71         /// <remarks>Используется в функции обработчике, который передаётся в функцию
72         ↪ Each.</remarks>
73         public TLinkAddress Break
74         {
75             [MethodImpl(MethodImplOptions.AggressiveInlining)]
76             get;
77         }
78
79         #endregion
80     }
81 }

```

```

74 #region Special symbols
75
76
77 /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
78 public TLinkAddress Null
79 {
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     get;
82 }
83
84 /// <summary>Возвращает значение, обозначающее любую связь.</summary>
85 /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
86   ↳ создавать все варианты последовательностей в функции Create.</remarks>
87 public TLinkAddress Any
88 {
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     get;
91 }
92
93 /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
94 public TLinkAddress Itself
95 {
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     get;
98 }
99 #endregion
100
101 #region References
102
103 /// <summary>Возвращает диапазон возможных индексов для внутренних связей (внутренних
104   ↳ ссылок).</summary>
105 public Range<TLinkAddress> InternalReferencesRange
106 {
107     [MethodImpl(MethodImplOptions.AggressiveInlining)]
108     get;
109 }
110
111 /// <summary>Возвращает диапазон возможных индексов для внешних связей (внешних
112   ↳ ссылок).</summary>
113 public Range<TLinkAddress>? ExternalReferencesRange
114 {
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     get;
117 }
118 #endregion
119
120 /// <summary>
121 /// <para>
122 /// Initializes a new <see cref="LinksConstants"/> instance.
123 /// </para>
124 /// </summary>
125 /// <param name="targetPart">
126 /// <para>A target part.</para>
127 /// </param>
128 /// <param name="possibleInternalReferencesRange">
129 /// <para>A possible internal references range.</para>
130 /// </param>
131 /// <param name="possibleExternalReferencesRange">
132 /// <para>A possible external references range.</para>
133 /// </param>
134 [MethodImpl(MethodImplOptions.AggressiveInlining)]
135 public LinksConstants(int targetPart, Range<TLinkAddress>
136   ↳ possibleInternalReferencesRange, Range<TLinkAddress>?
137   ↳ possibleExternalReferencesRange)
138 {
139     IndexPart = 0;
140     SourcePart = 1;
141     TargetPart = targetPart;
142     Null = default;
143     Break = default;
144     var currentInternalReferenceIndex = possibleInternalReferencesRange.Maximum;
145     Continue = currentInternalReferenceIndex;
146     Skip = Arithmetic.Decrement(ref currentInternalReferenceIndex);
147     Any = Arithmetic.Decrement(ref currentInternalReferenceIndex);
148 }

```

```

149         Itself = Arithmetic.Decrement(ref currentInternalReferenceIndex);
150         Arithmetic.Decrement(ref currentInternalReferenceIndex);
151         InternalReferencesRange = (possibleInternalReferencesRange.Minimum,
            ↪ currentInternalReferenceIndex);
152         ExternalReferencesRange = possibleExternalReferencesRange;
153     }
154
155     /// <summary>
156     /// <para>
157     /// Initializes a new <see cref="LinksConstants"/> instance.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="targetPart">
162     /// <para>A target part.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="enableExternalReferencesSupport">
166     /// <para>A enable external references support.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     public LinksConstants(int targetPart, bool enableExternalReferencesSupport) :
        ↪ this(targetPart, GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
        ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
171
172     /// <summary>
173     /// <para>
174     /// Initializes a new <see cref="LinksConstants"/> instance.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <param name="possibleInternalReferencesRange">
179     /// <para>A possible internal references range.</para>
180     /// <para></para>
181     /// </param>
182     /// <param name="possibleExternalReferencesRange">
183     /// <para>A possible external references range.</para>
184     /// <para></para>
185     /// </param>
186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
187     public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange,
        ↪ Range<TLinkAddress>? possibleExternalReferencesRange) : this(DefaultTargetPart,
        ↪ possibleInternalReferencesRange, possibleExternalReferencesRange) { }
188
189     /// <summary>
190     /// <para>
191     /// Initializes a new <see cref="LinksConstants"/> instance.
192     /// </para>
193     /// <para></para>
194     /// </summary>
195     /// <param name="enableExternalReferencesSupport">
196     /// <para>A enable external references support.</para>
197     /// <para></para>
198     /// </param>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     public LinksConstants(bool enableExternalReferencesSupport) :
        ↪ this(GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
        ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
201
202     /// <summary>
203     /// <para>
204     /// Initializes a new <see cref="LinksConstants"/> instance.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="targetPart">
209     /// <para>A target part.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="possibleInternalReferencesRange">
213     /// <para>A possible internal references range.</para>
214     /// <para></para>
215     /// </param>
216     [MethodImpl(MethodImplOptions.AggressiveInlining)]
217     public LinksConstants(int targetPart, Range<TLinkAddress>
        ↪ possibleInternalReferencesRange) : this(targetPart, possibleInternalReferencesRange,
        ↪ null) { }

```

```

218
219 /// <summary>
220 /// <para>
221 /// Initializes a new <see cref="LinksConstants"/> instance.
222 /// </para>
223 /// <para></para>
224 /// </summary>
225 /// <param name="possibleInternalReferencesRange">
226 /// <para>A possible internal references range.</para>
227 /// <para></para>
228 /// </param>
229 [MethodImpl(MethodImplOptions.AggressiveInlining)]
230 public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange) :
    ↪ this(DefaultTargetPart, possibleInternalReferencesRange, null) { }
231
232 /// <summary>
233 /// <para>
234 /// Initializes a new <see cref="LinksConstants"/> instance.
235 /// </para>
236 /// <para></para>
237 /// </summary>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 public LinksConstants() : this(DefaultTargetPart, enableExternalReferencesSupport:
    ↪ false) { }
240
241 /// <summary>
242 /// <para>
243 /// Gets the default internal references range using the specified enable external
    ↪ references support.
244 /// </para>
245 /// <para></para>
246 /// </summary>
247 /// <param name="enableExternalReferencesSupport">
248 /// <para>The enable external references support.</para>
249 /// <para></para>
250 /// </param>
251 /// <returns>
252 /// <para>A range of t link address</para>
253 /// <para></para>
254 /// </returns>
255 [MethodImpl(MethodImplOptions.AggressiveInlining)]
256 public static Range<TLinkAddress> GetDefaultInternalReferencesRange(bool
    ↪ enableExternalReferencesSupport)
257 {
258     if (enableExternalReferencesSupport)
259     {
260         return (_one, _uInt64ToAddressConverter.Convert(Hybrid<TLinkAddress>.HalfOfNumbe
            ↪ rValuesRange));
261     }
262     else
263     {
264         return (_one, NumericType<TLinkAddress>.MaxValue);
265     }
266 }
267
268 /// <summary>
269 /// <para>
270 /// Gets the default external references range using the specified enable external
    ↪ references support.
271 /// </para>
272 /// <para></para>
273 /// </summary>
274 /// <param name="enableExternalReferencesSupport">
275 /// <para>The enable external references support.</para>
276 /// <para></para>
277 /// </param>
278 /// <returns>
279 /// <para>A range of t link address</para>
280 /// <para></para>
281 /// </returns>
282 [MethodImpl(MethodImplOptions.AggressiveInlining)]
283 public static Range<TLinkAddress>? GetDefaultExternalReferencesRange(bool
    ↪ enableExternalReferencesSupport)
284 {
285     if (enableExternalReferencesSupport)
286     {
287         return (Hybrid<TLinkAddress>.ExternalZero, NumericType<TLinkAddress>.MaxValue);

```

```

288     }
289     else
290     {
291         return null;
292     }
293 }
294 }
295 }

```

1.12 ./csharp/Platform.Data/LinksConstantsBase.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the links constants base.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public abstract class LinksConstantsBase
12     {
13         /// <summary>
14         /// <para>
15         /// The default target part.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         public static readonly int DefaultTargetPart = 2;
20     }
21 }

```

1.13 ./csharp/Platform.Data/LinksConstantsExtensions.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  using System.Runtime.CompilerServices;
4
5  namespace Platform.Data
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links constants extensions.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public static class LinksConstantsExtensions
14     {
15         /// <summary>
16         /// <para>
17         /// Determines whether is reference.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         /// <typeparam name="TLinkAddress">
22         /// <para>The link address.</para>
23         /// <para></para>
24         /// </typeparam>
25         /// <param name="linksConstants">
26         /// <para>The links constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="address">
30         /// <para>The address.</para>
31         /// <para></para>
32         /// </param>
33         /// <returns>
34         /// <para>The bool</para>
35         /// <para></para>
36         /// </returns>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public static bool IsReference<TLinkAddress>(this LinksConstants<TLinkAddress>
39             ↪ linksConstants, TLinkAddress address) => linksConstants.IsInternalReference(address)
40             ↪ || linksConstants.IsExternalReference(address);
41
42         /// <summary>
43         /// <para>
44         /// Determines whether is internal reference.
45         /// </para>

```



```

44     /// <para></para>
45     /// </summary>
46     /// <typeparam name="TLinkAddress">
47     /// <para>The link address.</para>
48     /// <para></para>
49     /// </typeparam>
50     /// <param name="linksConstants">
51     /// <para>The links constants.</para>
52     /// <para></para>
53     /// </param>
54     /// <param name="address">
55     /// <para>The address.</para>
56     /// <para></para>
57     /// </param>
58     /// <returns>
59     /// <para>The bool</para>
60     /// <para></para>
61     /// </returns>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public static bool IsInternalReference<TLinkAddress>(this LinksConstants<TLinkAddress>
        ↳ linksConstants, TLinkAddress address) =>
        ↳ linksConstants.InternalReferencesRange.Contains(address);
64
65     /// <summary>
66     /// <para>
67     /// Determines whether is external reference.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <typeparam name="TLinkAddress">
72     /// <para>The link address.</para>
73     /// <para></para>
74     /// </typeparam>
75     /// <param name="linksConstants">
76     /// <para>The links constants.</para>
77     /// <para></para>
78     /// </param>
79     /// <param name="address">
80     /// <para>The address.</para>
81     /// <para></para>
82     /// </param>
83     /// <returns>
84     /// <para>The bool</para>
85     /// <para></para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     public static bool IsExternalReference<TLinkAddress>(this LinksConstants<TLinkAddress>
        ↳ linksConstants, TLinkAddress address) =>
        ↳ linksConstants.ExternalReferencesRange?.Contains(address) ?? false;
89 }
90 }

```

1.14 ./csharp/Platform.Data/Numbers/Raw/AddressToRawNumberConverter.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Converters;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Numbers.Raw
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the address to raw number converter.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="IConverter{TLink}"/>
15     public class AddressToRawNumberConverter<TLink> : IConverter<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Converts the source.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="source">
24         /// <para>The source.</para>
25         /// <para></para>

```

```

26     /// </param>
27     /// <returns>
28     /// <para>The link</para>
29     /// <para></para>
30     /// </returns>
31     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32     public TLink Convert(TLink source) => new Hybrid<TLink>(source, isExternal: true);
33 }
34 }

```

1.15 ./csharp/Platform.Data/Numbers/Raw/RawNumberToAddressConverter.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Converters;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Numbers.Raw
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the raw number to address converter.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="IConverter{TLink}"/>
15    public class RawNumberToAddressConverter<TLink> : IConverter<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// The default.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        static private readonly UncheckedConverter<long, TLink> _converter =
24            ↪ UncheckedConverter<long, TLink>.Default;
25
26        /// <summary>
27        /// <para>
28        /// Converts the source.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        /// <param name="source">
33        /// <para>The source.</para>
34        /// <para></para>
35        /// </param>
36        /// <returns>
37        /// <para>The link</para>
38        /// <para></para>
39        /// </returns>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public TLink Convert(TLink source) => _converter.Convert(new
42            ↪ Hybrid<TLink>(source).AbsoluteValue);
43    }
44 }

```

1.16 ./csharp/Platform.Data/Point.cs

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Runtime.CompilerServices;
5 using Platform.Exceptions;
6 using Platform.Ranges;
7 using Platform.Collections;
8
9 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the point.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="IEquatable{LinkAddress{TLinkAddress}}"/>
20     /// <seealso cref="IList{TLinkAddress}"/>
21     public class Point<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>, IList<TLinkAddress>
22     {

```

```

23 private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
24     ↳ EqualityComparer<TLinkAddress>.Default;
25
26 /// <summary>
27 /// <para>
28 /// Gets the index value.
29 /// </para>
30 /// <para></para>
31 /// </summary>
32 public TLinkAddress Index
33 {
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     get;
36 }
37
38 /// <summary>
39 /// <para>
40 /// Gets the size value.
41 /// </para>
42 /// <para></para>
43 /// </summary>
44 public int Size
45 {
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     get;
48 }
49
50 /// <summary>
51 /// <para>
52 /// The not supported exception.
53 /// </para>
54 /// <para></para>
55 /// </summary>
56 public TLinkAddress this[int index]
57 {
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     get
60     {
61         if (index < Size)
62         {
63             return Index;
64         }
65         else
66         {
67             throw new IndexOutOfRangeException();
68         }
69     }
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     set => throw new NotSupportedException();
72 }
73
74 /// <summary>
75 /// <para>
76 /// Gets the count value.
77 /// </para>
78 /// <para></para>
79 /// </summary>
80 public int Count
81 {
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     get => Size;
84 }
85
86 /// <summary>
87 /// <para>
88 /// Gets the is read only value.
89 /// </para>
90 /// <para></para>
91 /// </summary>
92 public bool IsReadOnly
93 {
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     get => true;
96 }
97
98 /// <summary>
99 /// <para>
100 /// Initializes a new <see cref="Point"/> instance.
101 /// </para>

```

```

101     /// <para></para>
102     /// </summary>
103     /// <param name="index">
104     /// <para>A index.</para>
105     /// <para></para>
106     /// </param>
107     /// <param name="size">
108     /// <para>A size.</para>
109     /// <para></para>
110     /// </param>
111     [MethodImpl(MethodImplOptions.AggressiveInlining)]
112     public Point(TLinkAddress index, int size)
113     {
114         Index = index;
115         Size = size;
116     }
117
118     /// <summary>
119     /// <para>
120     /// Adds the item.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="item">
125     /// <para>The item.</para>
126     /// <para></para>
127     /// </param>
128     [MethodImpl(MethodImplOptions.AggressiveInlining)]
129     public void Add(TLinkAddress item) => throw new NotSupportedException();
130
131     /// <summary>
132     /// <para>
133     /// Clears this instance.
134     /// </para>
135     /// <para></para>
136     /// </summary>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public void Clear() => throw new NotSupportedException();
139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance contains.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="item">
147     /// <para>The item.</para>
148     /// <para></para>
149     /// </param>
150     /// <returns>
151     /// <para>The bool</para>
152     /// <para></para>
153     /// </returns>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     public virtual bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index)
156     ↪ ? true : false;
157
158     /// <summary>
159     /// <para>
160     /// Copies the to using the specified array.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="array">
165     /// <para>The array.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="arrayIndex">
169     /// <para>The array index.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
174
175     /// <summary>
176     /// <para>
177     /// Gets the enumerator.
178     /// </para>

```

```

178     /// <para></para>
179     /// </summary>
180     /// <returns>
181     /// <para>An enumerator of t link address</para>
182     /// <para></para>
183     /// </returns>
184     [MethodImpl(MethodImplOptions.AggressiveInlining)]
185     public IEnumerator<TLinkAddress> GetEnumerator()
186     {
187         for (int i = 0; i < Size; i++)
188         {
189             yield return Index;
190         }
191     }
192
193     /// <summary>
194     /// <para>
195     /// Indexes the of using the specified item.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="item">
200     /// <para>The item.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The int</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
209     ↪ 0 : -1;
210
211     /// <summary>
212     /// <para>
213     /// Inserts the index.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="index">
218     /// <para>The index.</para>
219     /// <para></para>
220     /// </param>
221     /// <param name="item">
222     /// <para>The item.</para>
223     /// <para></para>
224     /// </param>
225     [MethodImpl(MethodImplOptions.AggressiveInlining)]
226     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
227
228     /// <summary>
229     /// <para>
230     /// Determines whether this instance remove.
231     /// </para>
232     /// <para></para>
233     /// </summary>
234     /// <param name="item">
235     /// <para>The item.</para>
236     /// <para></para>
237     /// </param>
238     /// <returns>
239     /// <para>The bool</para>
240     /// <para></para>
241     /// </returns>
242     [MethodImpl(MethodImplOptions.AggressiveInlining)]
243     public bool Remove(TLinkAddress item) => throw new NotSupportedException();
244
245     /// <summary>
246     /// <para>
247     /// Removes the at using the specified index.
248     /// </para>
249     /// <para></para>
250     /// </summary>
251     /// <param name="index">
252     /// <para>The index.</para>
253     /// <para></para>
254     /// </param>
255     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

255 public void RemoveAt(int index) => throw new NotSupportedException();
256
257 /// <summary>
258 /// <para>
259 /// Gets the enumerator.
260 /// </para>
261 /// <para></para>
262 /// </summary>
263 /// <returns>
264 /// <para>The enumerator</para>
265 /// <para></para>
266 /// </returns>
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 IEnumerator IEnumerable.GetEnumerator()
269 {
270     for (int i = 0; i < Size; i++)
271     {
272         yield return Index;
273     }
274 }
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance equals.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="other">
283 /// <para>The other.</para>
284 /// <para></para>
285 /// </param>
286 /// <returns>
287 /// <para>The bool</para>
288 /// <para></para>
289 /// </returns>
290 [MethodImpl(MethodImplOptions.AggressiveInlining)]
291 public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
    ↪ _equalityComparer.Equals(Index, other.Index);
292
293 [MethodImpl(MethodImplOptions.AggressiveInlining)]
294 public static implicit operator TLinkAddress(Point<TLinkAddress> linkAddress) =>
    ↪ linkAddress.Index;
295
296 /// <summary>
297 /// <para>
298 /// Determines whether this instance equals.
299 /// </para>
300 /// <para></para>
301 /// </summary>
302 /// <param name="obj">
303 /// <para>The obj.</para>
304 /// <para></para>
305 /// </param>
306 /// <returns>
307 /// <para>The bool</para>
308 /// <para></para>
309 /// </returns>
310 [MethodImpl(MethodImplOptions.AggressiveInlining)]
311 public override bool Equals(object obj) => obj is Point<TLinkAddress> linkAddress ?
    ↪ Equals(linkAddress) : false;
312
313 /// <summary>
314 /// <para>
315 /// Gets the hash code.
316 /// </para>
317 /// <para></para>
318 /// </summary>
319 /// <returns>
320 /// <para>The int</para>
321 /// <para></para>
322 /// </returns>
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 public override int GetHashCode() => Index.GetHashCode();
325
326 /// <summary>
327 /// <para>
328 /// Returns the string.
329 /// </para>

```

```

330    /// <para></para>
331    /// </summary>
332    /// <returns>
333    /// <para>The string</para>
334    /// <para></para>
335    /// </returns>
336    [MethodImpl(MethodImplOptions.AggressiveInlining)]
337    public override string ToString() => Index.ToString();
338
339    [MethodImpl(MethodImplOptions.AggressiveInlining)]
340    public static bool operator ==(Point<TLinkAddress> left, Point<TLinkAddress> right)
341    {
342        if (left == null && right == null)
343        {
344            return true;
345        }
346        if (left == null)
347        {
348            return false;
349        }
350        return left.Equals(right);
351    }
352
353    [MethodImpl(MethodImplOptions.AggressiveInlining)]
354    public static bool operator !=(Point<TLinkAddress> left, Point<TLinkAddress> right) =>
355        ↪ !(left == right);
356
357    /// <summary>
358    /// <para>
359    /// Determines whether is full point.
360    /// </para>
361    /// </summary>
362    /// <param name="link">
363    /// <para>The link.</para>
364    /// <para></para>
365    /// </param>
366    /// <returns>
367    /// <para>The bool</para>
368    /// <para></para>
369    /// </returns>
370    [MethodImpl(MethodImplOptions.AggressiveInlining)]
371    public static bool IsFullPoint(params TLinkAddress[] link) =>
372        ↪ IsFullPoint((IList<TLinkAddress>)link);
373
374    /// <summary>
375    /// <para>
376    /// Determines whether is full point.
377    /// </para>
378    /// </summary>
379    /// <param name="link">
380    /// <para>The link.</para>
381    /// <para></para>
382    /// </param>
383    /// <returns>
384    /// <para>The bool</para>
385    /// <para></para>
386    /// </returns>
387    [MethodImpl(MethodImplOptions.AggressiveInlining)]
388    public static bool IsFullPoint(IList<TLinkAddress> link)
389    {
390        Ensure.Always.ArgumentNotEmpty(link, nameof(link));
391        Ensure.Always.ArgumentInRange(link.Count, (2, int.MaxValue), nameof(link), "Cannot
392        ↪ determine link's pointness using only its identifier.");
393        return IsFullPointUnchecked(link);
394    }
395
396    /// <summary>
397    /// <para>
398    /// Determines whether is full point unchecked.
399    /// </para>
400    /// </summary>
401    /// <param name="link">
402    /// <para>The link.</para>
403    /// <para></para>
404    /// </param>

```

```

405 /// <returns>
406 /// <para>The result.</para>
407 /// <para></para>
408 /// </returns>
409 [MethodImpl(MethodImplOptions.AggressiveInlining)]
410 public static bool IsFullPointUnchecked(ICollection<TLinkAddress> link)
411 {
412     var result = true;
413     for (var i = 1; result && i < link.Count; i++)
414     {
415         result = _equalityComparer.Equals(link[0], link[i]);
416     }
417     return result;
418 }
419
420 /// <summary>
421 /// <para>
422 /// Determines whether is partial point.
423 /// </para>
424 /// <para></para>
425 /// </summary>
426 /// <param name="link">
427 /// <para>The link.</para>
428 /// <para></para>
429 /// </param>
430 /// <returns>
431 /// <para>The bool</para>
432 /// <para></para>
433 /// </returns>
434 [MethodImpl(MethodImplOptions.AggressiveInlining)]
435 public static bool IsPartialPoint(params TLinkAddress[] link) =>
436     IsPartialPoint((ICollection<TLinkAddress>)link);
437
438 /// <summary>
439 /// <para>
440 /// Determines whether is partial point.
441 /// </para>
442 /// <para></para>
443 /// </summary>
444 /// <param name="link">
445 /// <para>The link.</para>
446 /// <para></para>
447 /// </param>
448 /// <returns>
449 /// <para>The bool</para>
450 /// <para></para>
451 /// </returns>
452 [MethodImpl(MethodImplOptions.AggressiveInlining)]
453 public static bool IsPartialPoint(ICollection<TLinkAddress> link)
454 {
455     Ensure.Always.ArgumentNotEmpty(link, nameof(link));
456     Ensure.Always.ArgumentInRange(link.Count, (2, int.MaxValue), nameof(link), "Cannot
457         ↪ determine link's pointness using only its identifier.");
458     return IsPartialPointUnchecked(link);
459 }
460
461 /// <summary>
462 /// <para>
463 /// Determines whether is partial point unchecked.
464 /// </para>
465 /// <para></para>
466 /// </summary>
467 /// <param name="link">
468 /// <para>The link.</para>
469 /// <para></para>
470 /// </param>
471 /// <returns>
472 /// <para>The result.</para>
473 /// <para></para>
474 /// </returns>
475 [MethodImpl(MethodImplOptions.AggressiveInlining)]
476 public static bool IsPartialPointUnchecked(ICollection<TLinkAddress> link)
477 {
478     var result = false;
479     for (var i = 1; !result && i < link.Count; i++)
480     {
481         result = _equalityComparer.Equals(link[0], link[i]);
482     }
483 }

```



```

481         return result;
482     }
483 }
484 }

```

1.17 ./csharp/Platform.Data/Universal/IUniLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  // ReSharper disable TypeParameterCanBeVariant
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Universal
8  {
9      /// <remarks>Minimal sufficient universal Links API (for bulk operations).</remarks>
10     public partial interface IUniLinks<TLinkAddress>
11     {
12         /// <summary>
13         /// <para>
14         /// Triggers the condition.
15         /// </para>
16         /// <para></para>
17         /// </summary>
18         /// <param name="condition">
19         /// <para>The condition.</para>
20         /// <para></para>
21         /// </param>
22         /// <param name="substitution">
23         /// <para>The substitution.</para>
24         /// <para></para>
25         /// </param>
26         /// <returns>
27         /// <para>A list of i list i list t link address</para>
28         /// <para></para>
29         /// </returns>
30         IList<IList<IList<TLinkAddress>>> Trigger(IList<TLinkAddress> condition,
31             ↳ IList<TLinkAddress> substitution);
32     }
33
34     /// <remarks>Minimal sufficient universal Links API (for step by step operations).</remarks>
35     public partial interface IUniLinks<TLinkAddress>
36     {
37         /// <returns>
38         /// TLinkAddress that represents True (was finished fully) or TLinkAddress that
39         /// ↳ represents False (was stopped).
40         /// This is done to assure ability to push up stop signal through recursion stack.
41         /// </returns>
42         /// <remarks>
43         /// { 0, 0, 0 } => { itself, itself, itself } // create
44         /// { 1, any, any } => { itself, any, 3 } // update
45         /// { 3, any, any } => { 0, 0, 0 } // delete
46         /// </remarks>
47         TLinkAddress Trigger(IList<TLinkAddress> patternOrCondition, Func<IList<TLinkAddress>,
48             ↳ TLinkAddress> matchHandler,
49             IList<TLinkAddress> substitution, Func<IList<TLinkAddress>,
50             ↳ IList<TLinkAddress>, TLinkAddress> substitutionHandler);
51
52         /// <summary>
53         /// <para>
54         /// Triggers the restriction.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="restriction">
59         /// <para>The restriction.</para>
60         /// <para></para>
61         /// </param>
62         /// <param name="matchedHandler">
63         /// <para>The matched handler.</para>
64         /// <para></para>
65         /// </param>
66         /// <param name="substitution">
67         /// <para>The substitution.</para>
68         /// <para></para>
69         /// </param>
70         /// <param name="substitutedHandler">
71         /// <para>The substituted handler.</para>
72         /// <para></para>
73         /// </param>

```

```

69     /// </param>
70     /// <returns>
71     /// <para>The link address</para>
72     /// <para></para>
73     /// </returns>
74     TLinkAddress Trigger(IList<TLinkAddress> restriction, Func<IList<TLinkAddress>,
    ↪     IList<TLinkAddress>, TLinkAddress> matchedHandler,
75         IList<TLinkAddress> substitution, Func<IList<TLinkAddress>, IList<TLinkAddress>,
    ↪     TLinkAddress> substitutedHandler);
76 }
77
78 /// <remarks>Extended with small optimization.</remarks>
79 public partial interface IUniLinks<TLinkAddress>
80 {
81     /// <remarks>
82     /// Something simple should be simple and optimized.
83     /// </remarks>
84     TLinkAddress Count(IList<TLinkAddress> restrictions);
85 }
86 }

```

1.18 ./csharp/Platform.Data/Universal/IUniLinksCRUD.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  // ReSharper disable TypeParameterCanBeVariant
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Universal
8  {
9      /// <remarks>
10     /// CRUD aliases for IUniLinks.
11     /// </remarks>
12     public interface IUniLinksCRUD<TLinkAddress>
13     {
14         /// <summary>
15         /// <para>
16         /// Reads the part type.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         /// <param name="partType">
21         /// <para>The part type.</para>
22         /// <para></para>
23         /// </param>
24         /// <param name="link">
25         /// <para>The link.</para>
26         /// <para></para>
27         /// </param>
28         /// <returns>
29         /// <para>The link address</para>
30         /// <para></para>
31         /// </returns>
32         TLinkAddress Read(int partType, TLinkAddress link);
33         /// <summary>
34         /// <para>
35         /// Reads the handler.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="handler">
40         /// <para>The handler.</para>
41         /// <para></para>
42         /// </param>
43         /// <param name="pattern">
44         /// <para>The pattern.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>The link address</para>
49         /// <para></para>
50         /// </returns>
51         TLinkAddress Read(Func<TLinkAddress, bool> handler, IList<TLinkAddress> pattern);
52         /// <summary>
53         /// <para>
54         /// Creates the parts.
55         /// </para>
56         /// <para></para>

```

```

57     /// </summary>
58     /// <param name="parts">
59     /// <para>The parts.</para>
60     /// <para></para>
61     /// </param>
62     /// <returns>
63     /// <para>The link address</para>
64     /// <para></para>
65     /// </returns>
66     TLinkAddress Create(IList<TLinkAddress> parts);
67     /// <summary>
68     /// <para>
69     /// Updates the before.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="before">
74     /// <para>The before.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="after">
78     /// <para>The after.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link address</para>
83     /// <para></para>
84     /// </returns>
85     TLinkAddress Update(IList<TLinkAddress> before, IList<TLinkAddress> after);
86     /// <summary>
87     /// <para>
88     /// Deletes the parts.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="parts">
93     /// <para>The parts.</para>
94     /// <para></para>
95     /// </param>
96     void Delete(IList<TLinkAddress> parts);
97 }
98 }

```

1.19 ./csharp/Platform.Data/Universal/IUniLinksGS.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  // ReSharper disable TypeParameterCanBeVariant
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Universal
8  {
9      /// <remarks>
10     /// Get/Set aliases for IUniLinks.
11     /// </remarks>
12     public interface IUniLinksGS<TLinkAddress>
13     {
14         /// <summary>
15         /// <para>
16         /// Gets the part type.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         /// <param name="partType">
21         /// <para>The part type.</para>
22         /// <para></para>
23         /// </param>
24         /// <param name="link">
25         /// <para>The link.</para>
26         /// <para></para>
27         /// </param>
28         /// <returns>
29         /// <para>The link address</para>
30         /// <para></para>
31         /// </returns>
32         TLinkAddress Get(int partType, TLinkAddress link);
33         /// <summary>
34         /// <para>

```

```

35     /// Gets the handler.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="handler">
40     /// <para>The handler.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="pattern">
44     /// <para>The pattern.</para>
45     /// <para></para>
46     /// </param>
47     /// <returns>
48     /// <para>The link address</para>
49     /// <para></para>
50     /// </returns>
51     TLinkAddress Get(Func<TLinkAddress, bool> handler, IList<TLinkAddress> pattern);
52     /// <summary>
53     /// <para>
54     /// Sets the before.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="before">
59     /// <para>The before.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="after">
63     /// <para>The after.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link address</para>
68     /// <para></para>
69     /// </returns>
70     TLinkAddress Set(IList<TLinkAddress> before, IList<TLinkAddress> after);
71 }
72 }

```

1.20 ./csharp/Platform.Data/Universal/IUniLinksIO.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  // ReSharper disable TypeParameterCanBeVariant
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Universal
8  {
9      /// <remarks>
10     /// In/Out aliases for IUniLinks.
11     /// TLinkAddress can be any number type of any size.
12     /// </remarks>
13     public interface IUniLinksIO<TLinkAddress>
14     {
15         /// <remarks>
16         /// default(TLinkAddress) means any link.
17         /// Single element pattern means just element (link).
18         /// Handler gets array of link contents.
19         /// * link[0] is index or identifier.
20         /// * link[1] is source or first.
21         /// * link[2] is target or second.
22         /// * link[3] is linker or third.
23         /// * link[n] is nth part/parent/element/value
24         /// of link (if variable length links used).
25         ///
26         /// Stops and returns false if handler return false.
27         ///
28         /// Acts as Each, Foreach, Select, Search, Match & ...
29         ///
30         /// Handles all links in store if pattern/restrictions is not defined.
31         /// </remarks>
32         bool Out(Func<IList<TLinkAddress>, bool> handler, IList<TLinkAddress> pattern);
33
34         /// <remarks>
35         /// default(TLinkAddress) means itself.
36         /// Equivalent to:
37         /// * creation if before == null
38         /// * deletion if after == null

```

```

39     /// * update if before != null && after != null
40     /// * default(TLinkAddress) if before == null && after == null
41     ///
42     /// Possible interpretation
43     /// * In(null, new[] { }) creates point (link that points to itself using minimum number
44     ///   ↪ of parts).
45     /// * In(new[] { 4 }, null) deletes 4th link.
46     /// * In(new[] { 4 }, new [] { 5 }) delete 5th link if it exists and moves 4th link to
47     ///   ↪ 5th index.
48     /// * In(new[] { 4 }, new [] { 0, 2, 3 }) replaces 4th link with new doublet link (with
49     ///   ↪ 2 as source and 3 as target), 0 means it can be placed in any address.
50     /// ...
51     /// </remarks>
    TLinkAddress In(IList<TLinkAddress> before, IList<TLinkAddress> after);
}
}

```

1.21 ./csharp/Platform.Data/Universal/IUniLinksIOWithExtensions.cs

```

1  // ReSharper disable TypeParameterCanBeVariant
2  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
3
4  using System.Collections.Generic;
5
6  namespace Platform.Data.Universal
7  {
8      /// <remarks>Contains some optimizations of Out.</remarks>
9      public interface IUniLinksIOWithExtensions<TLinkAddress> : IUniLinksIO<TLinkAddress>
10     {
11         /// <remarks>
12         /// default(TLinkAddress) means nothing or null.
13         /// Single element pattern means just element (link).
14         /// OutPart(n, null) returns default(TLinkAddress).
15         /// OutPart(0, pattern) ~ Exists(link) or Search(pattern)
16         /// OutPart(1, pattern) ~ GetSource(link) or GetSource(Search(pattern))
17         /// OutPart(2, pattern) ~ GetTarget(link) or GetTarget(Search(pattern))
18         /// OutPart(3, pattern) ~ GetLinkAddresser(link) or GetLinkAddresser(Search(pattern))
19         /// OutPart(n, pattern) => For any variable length links, returns link or
20         ///   ↪ default(TLinkAddress).
21         ///
22         /// Outs(returns) inner contents of link, its part/parent/element/value.
23         /// </remarks>
24         TLinkAddress OutOne(int partType, IList<TLinkAddress> pattern);
25
26         /// <remarks>OutCount() returns total links in store as array.</remarks>
27         IList<IList<TLinkAddress>> OutAll(IList<TLinkAddress> pattern);
28
29         /// <remarks>OutCount() returns total amount of links in store.</remarks>
30         ulong OutCount(IList<TLinkAddress> pattern);
31     }
}

```

1.22 ./csharp/Platform.Data/Universal/IUniLinksRW.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  // ReSharper disable TypeParameterCanBeVariant
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Universal
8  {
9      /// <remarks>
10     /// Read/Write aliases for IUniLinks.
11     /// </remarks>
12     public interface IUniLinksRW<TLinkAddress>
13     {
14         /// <summary>
15         /// <para>
16         /// Reads the part type.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         /// <param name="partType">
21         /// <para>The part type.</para>
22         /// <para></para>
23         /// </param>
24         /// <param name="link">
25         /// <para>The link.</para>
26         /// <para></para>

```

```

27     /// </param>
28     /// <returns>
29     /// <para>The link address</para>
30     /// <para></para>
31     /// </returns>
32     TLinkAddress Read(int partType, TLinkAddress link);
33     /// <summary>
34     /// <para>
35     /// Determines whether this instance read.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="handler">
40     /// <para>The handler.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="pattern">
44     /// <para>The pattern.</para>
45     /// <para></para>
46     /// </param>
47     /// <returns>
48     /// <para>The bool</para>
49     /// <para></para>
50     /// </returns>
51     bool Read(Func<TLinkAddress, bool> handler, IList<TLinkAddress> pattern);
52     /// <summary>
53     /// <para>
54     /// Writes the before.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="before">
59     /// <para>The before.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="after">
63     /// <para>The after.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link address</para>
68     /// <para></para>
69     /// </returns>
70     TLinkAddress Write(IList<TLinkAddress> before, IList<TLinkAddress> after);
71 }
72 }

```

1.23 ./csharp/Platform.Data.Tests/HybridTests.cs

```

1  using Xunit;
2
3  namespace Platform.Data.Tests
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the hybrid tests.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public static class HybridTests
12     {
13         /// <summary>
14         /// <para>
15         /// Tests that object constructor test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void ObjectConstructorTest()
21         {
22             Assert.Equal(0, new Hybrid<byte>(unchecked((byte)128)).AbsoluteValue);
23             Assert.Equal(0, new Hybrid<byte>((object)128).AbsoluteValue);
24             Assert.Equal(1, new Hybrid<byte>(unchecked((byte)-1)).AbsoluteValue);
25             Assert.Equal(1, new Hybrid<byte>((object)-1).AbsoluteValue);
26             Assert.Equal(0, new Hybrid<byte>(unchecked((byte)0)).AbsoluteValue);
27             Assert.Equal(0, new Hybrid<byte>((object)0).AbsoluteValue);
28             Assert.Equal(1, new Hybrid<byte>(unchecked((byte)1)).AbsoluteValue);
29             Assert.Equal(1, new Hybrid<byte>((object)1).AbsoluteValue);
30         }
31     }
32 }

```

```
31     }
32 }
```

1.24 ./csharp/Platform.Data.Tests/LinksConstantsTests.cs

```
1  using Xunit;
2  using Platform.Reflection;
3  using Platform.Converters;
4  using Platform.Numbers;
5
6  namespace Platform.Data.Tests
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links constants tests.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     public static class LinksConstantsTests
15     {
16         /// <summary>
17         /// <para>
18         /// Tests that constructor test.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         [Fact]
23         public static void ConstructorTest()
24         {
25             var constants = new LinksConstants<ulong>(enableExternalReferencesSupport: true);
26             Assert.Equal(Hybrid<ulong>.ExternalZero,
27                 ↪ constants.ExternalReferencesRange.Value.Minimum);
28             Assert.Equal(ulong.MaxValue, constants.ExternalReferencesRange.Value.Maximum);
29
30             /// <summary>
31             /// <para>
32             /// Tests that external references test.
33             /// </para>
34             /// <para></para>
35             /// </summary>
36             [Fact]
37             public static void ExternalReferencesTest()
38             {
39                 TestExternalReferences<ulong, long>();
40                 TestExternalReferences<uint, int>();
41                 TestExternalReferences<ushort, short>();
42                 TestExternalReferences<byte, sbyte>();
43             }
44             private static void TestExternalReferences<TUnsigned, TSigned>()
45             {
46                 var unsingedOne = Arithmetic.Increment(default(TUnsigned));
47                 var converter = UncheckedConverter<TSigned, TUnsigned>.Default;
48                 var half = converter.Convert(NumericType<TSigned>.MaxValue);
49                 LinksConstants<TUnsigned> constants = new LinksConstants<TUnsigned>((unsingedOne,
50                     ↪ half), (Arithmetic.Add(half, unsingedOne), NumericType<TUnsigned>.MaxValue));
51
52                 var minimum = new Hybrid<TUnsigned>(default, isExternal: true);
53                 var maximum = new Hybrid<TUnsigned>(half, isExternal: true);
54
55                 Assert.True(constants.IsExternalReference(minimum));
56                 Assert.True(minimum.IsExternal);
57                 Assert.False(minimum.IsInternal);
58                 Assert.True(constants.IsExternalReference(maximum));
59                 Assert.True(maximum.IsExternal);
60                 Assert.False(maximum.IsInternal);
61             }
62     }
```

Index

- ./csharp/Platform.Data.Tests/HybridTests.cs, 38
- ./csharp/Platform.Data.Tests/LinksConstantsTests.cs, 39
- ./csharp/Platform.Data/Exceptions/ArgumentLinkDoesNotExistException.cs, 1
- ./csharp/Platform.Data/Exceptions/ArgumentLinkHasDependenciesException.cs, 2
- ./csharp/Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsException.cs, 3
- ./csharp/Platform.Data/Exceptions/LinksLimitReachedException.cs, 4
- ./csharp/Platform.Data/Exceptions/LinksLimitReachedExceptionBase.cs, 5
- ./csharp/Platform.Data/Hybrid.cs, 6
- ./csharp/Platform.Data/ILinks.cs, 10
- ./csharp/Platform.Data/ILinksExtensions.cs, 12
- ./csharp/Platform.Data/ISynchronizedLinks.cs, 15
- ./csharp/Platform.Data/LinkAddress.cs, 15
- ./csharp/Platform.Data/LinksConstants.cs, 19
- ./csharp/Platform.Data/LinksConstantsBase.cs, 24
- ./csharp/Platform.Data/LinksConstantsExtensions.cs, 24
- ./csharp/Platform.Data/Numbers/Raw/AddressToRawNumberConverter.cs, 25
- ./csharp/Platform.Data/Numbers/Raw/RawNumberToAddressConverter.cs, 26
- ./csharp/Platform.Data/Point.cs, 26
- ./csharp/Platform.Data/Universal/IUniLinks.cs, 33
- ./csharp/Platform.Data/Universal/IUniLinksCRUD.cs, 34
- ./csharp/Platform.Data/Universal/IUniLinksGS.cs, 35
- ./csharp/Platform.Data/Universal/IUniLinksIO.cs, 36
- ./csharp/Platform.Data/Universal/IUniLinksIOWithExtensions.cs, 37
- ./csharp/Platform.Data/Universal/IUniLinksRW.cs, 37