

# 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
86 /// <summary>
87 /// <para>
88 /// Formats the message using the specified link.
89 /// </para>
90 /// <para></para>
91 /// </summary>
92 /// <param name="link">
93 /// <para>The link.</para>
94 /// <para></para>
95 /// </param>
96 /// <param name="argumentName">
97 /// <para>The argument name.</para>
98 /// <para></para>
99 /// </param>
100 /// <returns>
101 /// <para>The string</para>
102 /// <para></para>
103 /// </returns>
104 [MethodImpl(MethodImplOptions.AggressiveInlining)]
105 private static string FormatMessage(TLinkAddress link, string argumentName) => $"Связь
    ↳ [{link}] переданная в аргумент [{argumentName}] не существует.";
106
107 /// <summary>
108 /// <para>
109 /// Formats the message using the specified link.
110 /// </para>
111 /// <para></para>
112 /// </summary>
113 /// <param name="link">
114 /// <para>The link.</para>
115 /// <para></para>
116 /// </param>
117 /// <returns>
118 /// <para>The string</para>
119 /// <para></para>
120 /// </returns>
121 [MethodImpl(MethodImplOptions.AggressiveInlining)]
122 private static string FormatMessage(TLinkAddress link) => $"Связь [{link}] переданная в
    ↳ качестве аргумента не существует.";
123 }
124 }

```

## 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>
59     /// <param name="innerException">
60     /// <para>A inner exception.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public ArgumentLinkHasDependenciesException(string message, Exception innerException) :
65         ↪ base(message, innerException) { }
66
67     /// <summary>
68     /// <para>
69     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="message">
74     /// <para>A message.</para>
75     /// <para></para>
76     /// </param>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public ArgumentLinkHasDependenciesException(string message) : base(message) { }
79
80     /// <summary>
81     /// <para>
82     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException"/> instance.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public ArgumentLinkHasDependenciesException() { }
88
89     /// <summary>
90     /// <para>
91     /// Formats the message using the specified link.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="link">
96     /// <para>The link.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="paramName">
100    /// <para>The param name.</para>
101    /// <para></para>

```

```

99     /// </param>
100    /// <returns>
101    /// <para>The string</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    private static string FormatMessage(TLinkAddress link, string paramName) => $"У связи
    ↳ [{link}] переданной в аргумент [{paramName}] присутствуют зависимости, которые
    ↳ препятствуют изменению её внутренней структуры.";

106
107    /// <summary>
108    /// <para>
109    /// Formats the message using the specified link.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="link">
114    /// <para>The link.</para>
115    /// <para></para>
116    /// </param>
117    /// <returns>
118    /// <para>The string</para>
119    /// <para></para>
120    /// </returns>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    private static string FormatMessage(TLinkAddress link) => $"У связи [{link}] переданной
    ↳ в качестве аргумента присутствуют зависимости, которые препятствуют изменению её
    ↳ внутренней структуры.";

123 }
124 }

```

### 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.

```

```

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     public LinkWithSameValueAlreadyExistsException(string message) : base(message) { }
54
55     /// <summary>
56     /// <para>
57     /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public LinkWithSameValueAlreadyExistsException() : base(DefaultMessage) { }
63 }
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) { }
47
48         /// <summary>
49         /// <para>
50         /// Initializes a new <see cref="LinksLimitReachedException"/> instance.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="message">
55         /// <para>A message.</para>
56         /// <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
69     /// <summary>
70     /// <para>
71     /// Formats the message using the specified limit.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <param name="limit">
76     /// <para>The limit.</para>
77     /// <para></para>
78     /// </param>
79     /// <returns>
80     /// <para>The string</para>
81     /// <para></para>
82     /// </returns>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     private static string FormatMessage(TLinkAddress limit) => $"Достигнут лимит количества
    ↳ связей в хранилище ({limit}).";
85 }
86 }

```

## 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         /// <summary>
22         /// <para>
23         /// The default.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
28             ↳ EqualityComparer<TLinkAddress>.Default;
29         /// <summary>
30         /// <para>
31         /// The default.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         private static readonly UncheckedSignExtendingConverter<TLinkAddress, long>
36             ↳ _addressToInt64Converter = UncheckedSignExtendingConverter<TLinkAddress,
37             ↳ long>.Default;
38         /// <summary>
39         /// <para>
40         /// The default.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
45             ↳ = UncheckedConverter<long, TLinkAddress>.Default;
46         /// <summary>
47         /// <para>
48         /// The default.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         private static readonly UncheckedConverter<TLinkAddress, ulong>
53             ↳ _addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
54         /// <summary>
55         /// <para>
56         /// The default.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         private static readonly UncheckedConverter<ulong, TLinkAddress>
61             ↳ _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
62         /// <summary>
63         /// <para>
64         /// The default.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68     }
69 }

```

```

59     /// </para>
60     /// <para></para>
61     /// </summary>
62     private static readonly UncheckedConverter<object, long> _objectToInt64Converter =
        ↳ UncheckedConverter<object, long>.Default;
63
64     /// <summary>
65     /// <para>
66     /// The max value.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     public static readonly ulong HalfOfNumberValuesRange =
        ↳ _addressToUInt64Converter.Convert(NumericType<TLinkAddress>.MaxValue) / 2;
71     /// <summary>
72     /// <para>
73     /// The half of number values range.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     public static readonly TLinkAddress ExternalZero =
        ↳ _uInt64ToAddressConverter.Convert(HalfOfNumberValuesRange + 1UL);
78
79     /// <summary>
80     /// <para>
81     /// The value.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     public readonly TLinkAddress Value;
86
87     /// <summary>
88     /// <para>
89     /// Gets the is nothing value.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     public bool IsNothing
94     {
95         [MethodImpl(MethodImplOptions.AggressiveInlining)]
96         get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue == 0;
97     }
98
99     /// <summary>
100    /// <para>
101    /// Gets the is internal value.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    public bool IsInternal
106    {
107        [MethodImpl(MethodImplOptions.AggressiveInlining)]
108        get => SignedValue > 0;
109    }
110
111    /// <summary>
112    /// <para>
113    /// Gets the is external value.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    public bool IsExternal
118    {
119        [MethodImpl(MethodImplOptions.AggressiveInlining)]
120        get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue < 0;
121    }
122
123    /// <summary>
124    /// <para>
125    /// Gets the signed value value.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    public long SignedValue
130    {
131        [MethodImpl(MethodImplOptions.AggressiveInlining)]
132        get => _addressToInt64Converter.Convert(Value);
133    }
134

```



```

135     /// <summary>
136     /// <para>
137     /// Gets the absolute value value.
138     /// </para>
139     /// <para></para>
140     /// </summary>
141     public long AbsoluteValue
142     {
143         [MethodImpl(MethodImplOptions.AggressiveInlining)]
144         get => _equalityComparer.Equals(Value, ExternalZero) ? 0 :
            ↳ Platform.Numbers.Math.Abs(SignedValue);
145     }
146
147     /// <summary>
148     /// <para>
149     /// Initializes a new <see cref="Hybrid"/> instance.
150     /// </para>
151     /// <para></para>
152     /// </summary>
153     /// <param name="value">
154     /// <para>A value.</para>
155     /// <para></para>
156     /// </param>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     public Hybrid(TLinkAddress value)
159     {
160         Ensure.OnDebug.IsUnsignedInteger<TLinkAddress>();
161         Value = value;
162     }
163
164     /// <summary>
165     /// <para>
166     /// Initializes a new <see cref="Hybrid"/> instance.
167     /// </para>
168     /// <para></para>
169     /// </summary>
170     /// <param name="value">
171     /// <para>A value.</para>
172     /// <para></para>
173     /// </param>
174     /// <param name="isExternal">
175     /// <para>A is external.</para>
176     /// <para></para>
177     /// </param>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     public Hybrid(TLinkAddress value, bool isExternal)
180     {
181         if (_equalityComparer.Equals(value, default) && isExternal)
182         {
183             Value = ExternalZero;
184         }
185         else
186         {
187             if (isExternal)
188             {
189                 Value = Math<TLinkAddress>.Negate(value);
190             }
191             else
192             {
193                 Value = value;
194             }
195         }
196     }
197
198     /// <summary>
199     /// <para>
200     /// Initializes a new <see cref="Hybrid"/> instance.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <param name="value">
205     /// <para>A value.</para>
206     /// <para></para>
207     /// </param>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     public Hybrid(object value) => Value =
        ↳ _int64ToAddressConverter.Convert(_objectToInt64Converter.Convert(value));
210

```

```

211 /// <summary>
212 /// <para>
213 /// Initializes a new <see cref="Hybrid"/> instance.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="value">
218 /// <para>A value.</para>
219 /// <para></para>
220 /// </param>
221 /// <param name="isExternal">
222 /// <para>A is external.</para>
223 /// <para></para>
224 /// </param>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 public Hybrid(object value, bool isExternal)
227 {
228     var signedValue = value == null ? 0 : _objectToInt64Converter.Convert(value);
229     if (signedValue == 0 && isExternal)
230     {
231         Value = ExternalZero;
232     }
233     else
234     {
235         var absoluteValue = System.Math.Abs(signedValue);
236         Value = isExternal ? _int64ToAddressConverter.Convert(-absoluteValue) :
                ↪ _int64ToAddressConverter.Convert(absoluteValue);
237     }
238 }
239
240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
241 public static implicit operator Hybrid(TLinkAddress integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
242
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 public static explicit operator Hybrid(TLinkAddress)(ulong integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
245
246 [MethodImpl(MethodImplOptions.AggressiveInlining)]
247 public static explicit operator Hybrid(TLinkAddress)(long integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
248
249 [MethodImpl(MethodImplOptions.AggressiveInlining)]
250 public static explicit operator Hybrid(TLinkAddress)(uint integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
251
252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
253 public static explicit operator Hybrid(TLinkAddress)(int integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
254
255 [MethodImpl(MethodImplOptions.AggressiveInlining)]
256 public static explicit operator Hybrid(TLinkAddress)(ushort integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
257
258 [MethodImpl(MethodImplOptions.AggressiveInlining)]
259 public static explicit operator Hybrid(TLinkAddress)(short integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
260
261 [MethodImpl(MethodImplOptions.AggressiveInlining)]
262 public static explicit operator Hybrid(TLinkAddress)(byte integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
263
264 [MethodImpl(MethodImplOptions.AggressiveInlining)]
265 public static explicit operator Hybrid(TLinkAddress)(sbyte integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
266
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 public static implicit operator TLinkAddress(Hybrid<TLinkAddress> hybrid) =>
    ↪ hybrid.Value;
269
270 [MethodImpl(MethodImplOptions.AggressiveInlining)]
271 public static explicit operator ulong(Hybrid<TLinkAddress> hybrid) =>
    ↪ CheckedConverter<TLinkAddress, ulong>.Default.Convert(hybrid.Value);
272
273 [MethodImpl(MethodImplOptions.AggressiveInlining)]
274 public static explicit operator long(Hybrid<TLinkAddress> hybrid) =>
    ↪ hybrid.AbsoluteValue;
275

```

```

276 [MethodImpl(MethodImplOptions.AggressiveInlining)]
277 public static explicit operator uint(Hybrid<TLinkAddress> hybrid) =>
    ↳ CheckedConverter<TLinkAddress, uint>.Default.Convert(hybrid.Value);
278
279 [MethodImpl(MethodImplOptions.AggressiveInlining)]
280 public static explicit operator int(Hybrid<TLinkAddress> hybrid) =>
    ↳ (int)hybrid.AbsoluteValue;
281
282 [MethodImpl(MethodImplOptions.AggressiveInlining)]
283 public static explicit operator ushort(Hybrid<TLinkAddress> hybrid) =>
    ↳ CheckedConverter<TLinkAddress, ushort>.Default.Convert(hybrid.Value);
284
285 [MethodImpl(MethodImplOptions.AggressiveInlining)]
286 public static explicit operator short(Hybrid<TLinkAddress> hybrid) =>
    ↳ (short)hybrid.AbsoluteValue;
287
288 [MethodImpl(MethodImplOptions.AggressiveInlining)]
289 public static explicit operator byte(Hybrid<TLinkAddress> hybrid) =>
    ↳ CheckedConverter<TLinkAddress, byte>.Default.Convert(hybrid.Value);
290
291 [MethodImpl(MethodImplOptions.AggressiveInlining)]
292 public static explicit operator sbyte(Hybrid<TLinkAddress> hybrid) =>
    ↳ (sbyte)hybrid.AbsoluteValue;
293
294 /// <summary>
295 /// <para>
296 /// Returns the string.
297 /// </para>
298 /// </summary>
299 /// <returns>
300 /// <para>The string</para>
301 /// </returns>
302 [MethodImpl(MethodImplOptions.AggressiveInlining)]
303 public override string ToString() => IsExternal ? $"{<AbsoluteValue>}" :
    ↳ Value.ToString();
304
305 /// <summary>
306 /// <para>
307 /// Determines whether this instance equals.
308 /// </para>
309 /// <param name="other">
310 /// <para>The other.</para>
311 /// </param>
312 /// <returns>
313 /// <para>The bool</para>
314 /// </returns>
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 public bool Equals(Hybrid<TLinkAddress> other) => _equalityComparer.Equals(Value,
    ↳ other.Value);
317
318 /// <summary>
319 /// <para>
320 /// Determines whether this instance equals.
321 /// </para>
322 /// <param name="obj">
323 /// <para>The obj.</para>
324 /// </param>
325 /// <returns>
326 /// <para>The bool</para>
327 /// </returns>
328 [MethodImpl(MethodImplOptions.AggressiveInlining)]
329 public override bool Equals(object obj) => obj is Hybrid<TLinkAddress> hybrid ?
    ↳ Equals(hybrid) : false;
330
331 /// <summary>
332 /// <para>
333 /// Gets the hash code.

```

```

344     /// </para>
345     /// <para></para>
346     /// </summary>
347     /// <returns>
348     /// <para>The int</para>
349     /// <para></para>
350     /// </returns>
351     [MethodImpl(MethodImplOptions.AggressiveInlining)]
352     public override int GetHashCode() => Value.GetHashCode();
353
354     [MethodImpl(MethodImplOptions.AggressiveInlining)]
355     public static bool operator ==(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
        ↳ left.Equals(right);
356
357     [MethodImpl(MethodImplOptions.AggressiveInlining)]
358     public static bool operator !=(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
        ↳ !(left == right);
359 }
360 }

```

## 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>
22     where TConstants : LinksConstants<TLinkAddress>
23     {
24         #region Constants
25
26         /// <summary>
27         /// <para>Returns the set of constants that is necessary for effective communication
28         ↳ with the methods of this interface.</para>
29         /// <para>Возвращает набор констант, который необходим для эффективной коммуникации с
30         ↳ методами этого интерфейса.</para>
31         /// </summary>
32         /// <remarks>
33         /// <para>These constants are not changed since the creation of the links storage access
34         ↳ point.</para>
35         /// <para>Эти константы не меняются с момента создания точки доступа к хранилищу
36         ↳ связей.</para>
37         /// </remarks>
38         TConstants Constants
39         {
40             [MethodImpl(MethodImplOptions.AggressiveInlining)]
41             get;
42         }
43
44         #endregion
45
46         #region Read
47
48         /// <summary>
49         /// <para>Counts and returns the total number of links in the storage that meet the
50         ↳ specified restrictions.</para>
51         /// <para>Подсчитывает и возвращает общее число связей находящихся в хранилище,
52         ↳ соответствующих указанным ограничениям.</para>
53         /// </summary>
54         /// <param name="restriction"><para>Restrictions on the contents of
55         ↳ links.</para><para>Ограничения на содержимое связей.</para></param>

```

```

45  /// <returns><para>The total number of links in the storage that meet the specified
    ↳ restrictions.</para><para>Общее число связей находящихся в хранилище,
    ↳ соответствующих указанным ограничениям.</para></returns>
46 [MethodImpl(MethodImplOptions.AggressiveInlining)]
47 TLinkAddress Count(IList<TLinkAddress> restriction);
48
49  /// <summary>
50  /// <para>Passes through all the links matching the pattern, invoking a handler for each
    ↳ matching link.</para>
51  /// <para>Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.</para>
52  /// </summary>
53  /// <param name="handler"><para>A handler for each matching link.</para><para>Обработчик
    ↳ для каждой подходящей связи.</para></param>
54  /// <param name="restrictions">
55  /// <para>Restrictions on the contents of links. Each constraint can have values:
    ↳ Constants.Null - the 0th link denoting a reference to the void, Any - the absence of
    ↳ a constraint, 1.. $\infty$  a specific link index.</para>
56  /// <para>Ограничения на содержимое связей. Каждое ограничение может иметь значения:
    ↳ Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any - отсутствие
    ↳ ограничения, 1.. $\infty$  конкретный индекс связи.</para>
57  /// </param>
58  /// <returns><para>Constants.Continue, if the pass through the links was not
    ↳ interrupted, and Constants.Break otherwise.</para><para>Constants.Continue, в случае
    ↳ если проход по связям не был прерван и Constants.Break в обратном
    ↳ случае.</para></returns>
59 [MethodImpl(MethodImplOptions.AggressiveInlining)]
60 TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler, IList<TLinkAddress>
    ↳ restrictions);
61
62 #endregion
63
64 #region Write
65
66  /// <summary>
67  /// <para>Creates a link.</para>
68  /// <para>Создаёт связь.</para>
69  /// <param name="restrictions">
70  /// <para>Restrictions on the content of a link. This argument is optional, if the null
    ↳ passed as value that means no restrictions on the content of a link are set.</para>
71  /// <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>
72  /// </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> restrictions); // TODO: Возвращать связь
    ↳ возвращать нужно целиком.
77
78  /// <summary>
79  /// Обновляет связь с указанными restrictions[Constants.IndexPart] в адресом связи
80  /// на связь с указанным новым содержимым.
81  /// </summary>
82  /// <param name="restrictions">
83  /// Ограничения на содержимое связей.
84  /// Предполагается, что будет указан индекс связи (в restrictions[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> restrictions, IList<TLinkAddress> substitution);
    ↳ // TODO: Возможно и возвращать связь нужно целиком.
92
93  /// <summary>
94  /// <para>Deletes links that match the specified restrictions.</para>
95  /// <para>Удаляет связи соответствующие указанным ограничениям.</para>
96  /// <param name="restrictions">
97  /// <para>Restrictions on the content of a link. This argument is optional, if the null
    ↳ passed as value that means no restrictions 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(ICollection<TLinkAddress> restrictions); // TODO: Возможно всегда нужно принимать
    ↪ restrictions, а так же возвращать удалённую связь, если удаление было реально
    ↪ выполнено, и 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)
        ↳ where TConstants : LinksConstants<TLinkAddress>
71     {
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)
        ↳ where TConstants : LinksConstants<TLinkAddress>
84     {
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
94     ↳ (handler) для каждой подходящей связи.
95     /// </summary>
96     /// <param name="links">Хранилище связей.</param>
97     /// <param name="handler">Обработчик каждой подходящей связи.</param>
98     /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
99     ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
100     ↳ Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
101     /// <returns>True, в случае если проход по связям не был прерван и False в обратном
102     ↳ случае.</returns>
103     [MethodImpl(MethodImplOptions.AggressiveInlining)]
104     public static TLinkAddress Each<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
        ↳ TConstants> links, Func<IList<TLinkAddress>, TLinkAddress> handler, params
        ↳ TLinkAddress[] restrictions)
        ↳ where TConstants : LinksConstants<TLinkAddress>
105     => links.Each(handler, restrictions);
106
107     /// <summary>
108     /// Возвращает части-значения для связи с указанным индексом.
109     /// </summary>
110     /// <param name="links">Хранилище связей.</param>
111     /// <param name="link">Индекс связи.</param>
112     /// <returns>Уникальную связь.</returns>
113     [MethodImpl(MethodImplOptions.AggressiveInlining)]
114     public static IList<TLinkAddress> GetLink<TLinkAddress, TConstants>(this
        ↳ ILinks<TLinkAddress, TConstants> links, TLinkAddress link)
        ↳ where TConstants : LinksConstants<TLinkAddress>
115     {
116     }
117     var constants = links.Constants;
118     if (constants.IsExternalReference(link))
119     {
120         return new Point<TLinkAddress>(link, constants.TargetPart + 1);
121     }
122     var linkPartsSetter = new Setter<IList<TLinkAddress>,
        ↳ TLinkAddress>(constants.Continue, constants.Break);
123     links.Each(linkPartsSetter.SetAndReturnTrue, link);
124     return linkPartsSetter.Result;
125 }
126
127 #region Points

```

```

126  /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
127  → точкой полностью (связью замкнутой на себе дважды).</summary>
128  /// <param name="links">Хранилище связей.</param>
129  /// <param name="link">Индекс проверяемой связи.</param>
130  /// <returns>Значение, определяющее является ли связь точкой полностью.</returns>
131  /// <remarks>
132  /// Связь точка - это связь, у которой начало (Source) и конец (Target) есть сама эта
133  → связь.
134  /// Но что, если точка уже есть, а нужно создать пару с таким же значением? Должны ли
135  → точка и пара существовать одновременно?
136  /// Или в качестве решения для точек нужно использовать 0 в качестве начала и конца, а
137  → сортировать по индексу в массиве связей?
138  /// Какое тогда будет значение Source и Target у точки? 0 или её индекс?
139  /// Или точка должна быть одновременно точкой и парой, а также последовательностями из
140  → самой себя любого размера?
141  /// Как только есть ссылка на себя, появляется этот парадокс, причём достаточно даже
142  → одной ссылки на себя (частичной точки).
143  /// А что если не выбирать что является точкой, пара нулей (цикл через пустоту) или
144  /// самостоятельный цикл через себя? Что если предоставить все варианты использования
145  → связей?
146  /// Что если разрешить и нули, а так же частичные варианты?
147  ///
148  /// Что если точка, это только в том случае когда link.Source == link &&
149  → link.Target == link , т.е. дважды ссылка на себя.
150  /// А пара это тогда, когда link.Source == link.Target && link.Source != link ,
151  → т.е. ссылка не на себя а во вне.
152  ///
153  /// Тогда если у нас уже создана пара, но нам нужна точка, мы можем используя
154  → промежуточную связь,
155  /// например "DoubletOf" обозначить что является точно парой, а что точно точкой.
156  /// И наоборот этот же метод поможет, если уже существует точка, но нам нужна пара.
157  /// </remarks>
158  [MethodImpl(MethodImplOptions.AggressiveInlining)]
159  public static bool IsFullPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
160  → TConstants> links, TLinkAddress link)
161  where TConstants : LinksConstants<TLinkAddress>
162  {
163  if (links.Constants.IsExternalReference(link))
164  {
165  return true;
166  }
167  links.EnsureLinkExists(link);
168  return Point<TLinkAddress>.IsFullPoint(links.GetLink(link));
169  }
170
171  /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
172  → точкой частично (связью замкнутой на себе как минимум один раз).</summary>
173  /// <param name="links">Хранилище связей.</param>
174  /// <param name="link">Индекс проверяемой связи.</param>
175  /// <returns>Значение, определяющее является ли связь точкой частично.</returns>
176  /// <remarks>
177  /// Достаточно любой одной ссылки на себя.
178  /// Также в будущем можно будет проверять и всех родителей, чтобы проверить есть ли
179  → ссылки на себя (на эту связь).
180  /// </remarks>
181  [MethodImpl(MethodImplOptions.AggressiveInlining)]
182  public static bool IsPartialPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
183  → TConstants> links, TLinkAddress link)
184  where TConstants : LinksConstants<TLinkAddress>
185  {
186  if (links.Constants.IsExternalReference(link))
187  {
188  return true;
189  }
190  links.EnsureLinkExists(link);
191  return Point<TLinkAddress>.IsPartialPoint(links.GetLink(link));
192  }
193
194  #endregion
195  }
196  }

```

## 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> :
16        ↳ ISynchronized<TLinks>, ILinks<TLinkAddress, TConstants>
17        where TLinks : ILinks<TLinkAddress, TConstants>
18        where TConstants : LinksConstants<TLinkAddress>
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>>,
19         ↳ IList<TLinkAddress>
20     {
21         /// <summary>
22         /// <para>
23         /// The default.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
28             ↳ EqualityComparer<TLinkAddress>.Default;
29
30         /// <summary>
31         /// <para>
32         /// Gets the index value.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         public TLinkAddress Index
37         {
38             [MethodImpl(MethodImplOptions.AggressiveInlining)]
39             get;
40         }
41
42         /// <summary>
43         /// <para>
44         /// The not supported exception.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         public TLinkAddress this[int index]
49         {
50             [MethodImpl(MethodImplOptions.AggressiveInlining)]
51             get
52             {
53                 if (index == 0)
54                 {
55                     return Index;
56                 }
57                 else
58                 {
59                     throw new IndexOutOfRangeException();
60                 }
61             }
62         }
63     }

```

```

60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         set => throw new NotSupportedException();
62     }
63
64     /// <summary>
65     /// <para>
66     /// Gets the count value.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     public int Count
71     {
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         get => 1;
74     }
75
76     /// <summary>
77     /// <para>
78     /// Gets the is read only value.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     public bool IsReadOnly
83     {
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]
85         get => true;
86     }
87
88     /// <summary>
89     /// <para>
90     /// Initializes a new <see cref="LinkAddress"/> instance.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="index">
95     /// <para>A index.</para>
96     /// <para></para>
97     /// </param>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public LinkAddress(TLinkAddress index) => Index = index;
100
101     /// <summary>
102     /// <para>
103     /// Adds the item.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="item">
108     /// <para>The item.</para>
109     /// <para></para>
110     /// </param>
111     [MethodImpl(MethodImplOptions.AggressiveInlining)]
112     public void Add(TLinkAddress item) => throw new NotSupportedException();
113
114     /// <summary>
115     /// <para>
116     /// Clears this instance.
117     /// </para>
118     /// <para></para>
119     /// </summary>
120     [MethodImpl(MethodImplOptions.AggressiveInlining)]
121     public void Clear() => throw new NotSupportedException();
122
123     /// <summary>
124     /// <para>
125     /// Determines whether this instance contains.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="item">
130     /// <para>The item.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The bool</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

138     public virtual bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index)
139         ↪ ? true : false;
140
141     /// <summary>
142     /// <para>
143     /// Copies the to using the specified array.
144     /// </para>
145     /// </summary>
146     /// <param name="array">
147     /// <para>The array.</para>
148     /// </param>
149     /// <param name="arrayIndex">
150     /// <para>The array index.</para>
151     /// </param>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
154
155     /// <summary>
156     /// <para>
157     /// Gets the enumerator.
158     /// </para>
159     /// </summary>
160     /// <returns>
161     /// <para>An enumerator of t link address</para>
162     /// </returns>
163     [MethodImpl(MethodImplOptions.AggressiveInlining)]
164     public IEnumerator<TLinkAddress> GetEnumerator()
165     {
166         yield return Index;
167     }
168
169     /// <summary>
170     /// <para>
171     /// Indexes the of using the specified item.
172     /// </para>
173     /// </summary>
174     /// <param name="item">
175     /// <para>The item.</para>
176     /// </param>
177     /// <returns>
178     /// <para>The int</para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
182         ↪ 0 : -1;
183
184     /// <summary>
185     /// <para>
186     /// Inserts the index.
187     /// </para>
188     /// </summary>
189     /// <param name="index">
190     /// <para>The index.</para>
191     /// </param>
192     /// <param name="item">
193     /// <para>The item.</para>
194     /// </param>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
197
198     /// <summary>
199     /// <para>
200     /// Determines whether this instance remove.
201     /// </para>
202     /// </summary>
203     /// <param name="item">

```

```

214    /// <para>The item.</para>
215    /// <para></para>
216    /// </param>
217    /// <returns>
218    /// <para>The bool</para>
219    /// <para></para>
220    /// </returns>
221    [MethodImpl(MethodImplOptions.AggressiveInlining)]
222    public bool Remove(TLinkAddress item) => throw new NotSupportedException();
223
224    /// <summary>
225    /// <para>
226    /// Removes the at using the specified index.
227    /// </para>
228    /// <para></para>
229    /// </summary>
230    /// <param name="index">
231    /// <para>The index.</para>
232    /// <para></para>
233    /// </param>
234    [MethodImpl(MethodImplOptions.AggressiveInlining)]
235    public void RemoveAt(int index) => throw new NotSupportedException();
236
237    /// <summary>
238    /// <para>
239    /// Gets the enumerator.
240    /// </para>
241    /// <para></para>
242    /// </summary>
243    /// <returns>
244    /// <para>The enumerator</para>
245    /// <para></para>
246    /// </returns>
247    [MethodImpl(MethodImplOptions.AggressiveInlining)]
248    IEnumerator IEnumerable.GetEnumerator()
249    {
250        yield return Index;
251    }
252
253    /// <summary>
254    /// <para>
255    /// Determines whether this instance equals.
256    /// </para>
257    /// <para></para>
258    /// </summary>
259    /// <param name="other">
260    /// <para>The other.</para>
261    /// <para></para>
262    /// </param>
263    /// <returns>
264    /// <para>The bool</para>
265    /// <para></para>
266    /// </returns>
267    [MethodImpl(MethodImplOptions.AggressiveInlining)]
268    public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
    ↪ _equalityComparer.Equals(Index, other.Index);
269
270    [MethodImpl(MethodImplOptions.AggressiveInlining)]
271    public static implicit operator TLinkAddress(LinkAddress<TLinkAddress> linkAddress) =>
    ↪ linkAddress.Index;
272
273    [MethodImpl(MethodImplOptions.AggressiveInlining)]
274    public static implicit operator LinkAddress<TLinkAddress>(TLinkAddress linkAddress) =>
    ↪ new LinkAddress<TLinkAddress>(linkAddress);
275
276    /// <summary>
277    /// <para>
278    /// Determines whether this instance equals.
279    /// </para>
280    /// <para></para>
281    /// </summary>
282    /// <param name="obj">
283    /// <para>The obj.</para>
284    /// <para></para>
285    /// </param>
286    /// <returns>
287    /// <para>The bool</para>
288    /// <para></para>

```

```

289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     public override bool Equals(object obj) => obj is LinkAddress<TLinkAddress> linkAddress
        ↪ ? Equals(linkAddress) : false;
292
293     /// <summary>
294     /// <para>
295     /// Gets the hash code.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <returns>
300     /// <para>The int</para>
301     /// <para></para>
302     /// </returns>
303     [MethodImpl(MethodImplOptions.AggressiveInlining)]
304     public override int GetHashCode() => Index.GetHashCode();
305
306     /// <summary>
307     /// <para>
308     /// Returns the string.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <returns>
313     /// <para>The string</para>
314     /// <para></para>
315     /// </returns>
316     [MethodImpl(MethodImplOptions.AggressiveInlining)]
317     public override string ToString() => Index.ToString();
318
319     [MethodImpl(MethodImplOptions.AggressiveInlining)]
320     public static bool operator ==(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
        ↪ right)
321     {
322         if (left == null && right == null)
323         {
324             return true;
325         }
326         if (left == null)
327         {
328             return false;
329         }
330         return left.Equals(right);
331     }
332
333     [MethodImpl(MethodImplOptions.AggressiveInlining)]
334     public static bool operator !=(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
        ↪ right) => !(left == right);
335 }
336 }

```

## 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         /// <summary>
21         /// <para>
22         /// The increment.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         private static readonly TLinkAddress _one = Arithmetic<TLinkAddress>.Increment(default);

```

```

27     /// <summary>
28     /// <para>
29     /// The default.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     private static readonly UncheckedConverter<ulong, TLinkAddress>
34     ↪ _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
35
36     #region Link parts
37
38     /// <summary>Возвращает индекс части, которая отвечает за индекс (адрес, идентификатор)
39     ↪ самой связи.</summary>
40     public int IndexPart
41     {
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         get;
44     }
45
46     /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-начало (первая
47     ↪ часть-значение).</summary>
48     public int SourcePart
49     {
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         get;
52     }
53
54     /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-конец
55     ↪ (последняя часть-значение).</summary>
56     public int TargetPart
57     {
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         get;
60     }
61
62     #endregion
63
64     #region Flow control
65
66     /// <summary>Возвращает значение, обозначающее продолжение прохода по связям.</summary>
67     /// <remarks>Используется в функции обработчике, который передаётся в функцию
68     ↪ Each.</remarks>
69     public TLinkAddress Continue
70     {
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         get;
73     }
74
75     /// <summary>Возвращает значение, обозначающее пропуск в проходе по связям.</summary>
76     public TLinkAddress Skip
77     {
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         get;
80     }
81
82     /// <summary>Возвращает значение, обозначающее остановку прохода по связям.</summary>
83     /// <remarks>Используется в функции обработчике, который передаётся в функцию
84     ↪ Each.</remarks>
85     public TLinkAddress Break
86     {
87         [MethodImpl(MethodImplOptions.AggressiveInlining)]
88         get;
89     }
90
91     #endregion
92
93     #region Special symbols
94
95     /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
96     public TLinkAddress Null
97     {
98         [MethodImpl(MethodImplOptions.AggressiveInlining)]
99         get;
100     }
101
102     /// <summary>Возвращает значение, обозначающее любую связь.</summary>
103     /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
104     ↪ создавать все варианты последовательностей в функции Create.</remarks>
105     public TLinkAddress Any
106     {
107

```

```

100     [MethodImpl(MethodImplOptions.AggressiveInlining)]
101     get;
102 }
103
104 /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
105 public TLinkAddress Itself
106 {
107     [MethodImpl(MethodImplOptions.AggressiveInlining)]
108     get;
109 }
110
111 #endregion
112
113 #region References
114
115 /// <summary>Возвращает диапазон возможных индексов для внутренних связей (внутренних
116 ↪ ссылок).</summary>
117 public Range<TLinkAddress> InternalReferencesRange
118 {
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     get;
121 }
122
123 /// <summary>Возвращает диапазон возможных индексов для внешних связей (внешних
124 ↪ ссылок).</summary>
125 public Range<TLinkAddress>? ExternalReferencesRange
126 {
127     [MethodImpl(MethodImplOptions.AggressiveInlining)]
128     get;
129 }
130
131 #endregion
132
133 /// <summary>
134 /// <para>
135 /// Initializes a new <see cref="LinksConstants"/> instance.
136 /// </para>
137 /// <para></para>
138 /// </summary>
139 /// <param name="targetPart">
140 /// <para>A target part.</para>
141 /// <para></para>
142 /// </param>
143 /// <param name="possibleInternalReferencesRange">
144 /// <para>A possible internal references range.</para>
145 /// <para></para>
146 /// </param>
147 /// <param name="possibleExternalReferencesRange">
148 /// <para>A possible external references range.</para>
149 /// <para></para>
150 /// </param>
151 [MethodImpl(MethodImplOptions.AggressiveInlining)]
152 public LinksConstants(int targetPart, Range<TLinkAddress>
153 ↪ possibleInternalReferencesRange, Range<TLinkAddress>?
154 ↪ possibleExternalReferencesRange)
155 {
156     IndexPart = 0;
157     SourcePart = 1;
158     TargetPart = targetPart;
159     Null = default;
160     Break = default;
161     var currentInternalReferenceIndex = possibleInternalReferencesRange.Maximum;
162     Continue = currentInternalReferenceIndex;
163     Skip = Arithmetic.Decrement(ref currentInternalReferenceIndex);
164     Any = Arithmetic.Decrement(ref currentInternalReferenceIndex);
165     Itself = Arithmetic.Decrement(ref currentInternalReferenceIndex);
166     Arithmetic.Decrement(ref currentInternalReferenceIndex);
167     InternalReferencesRange = (possibleInternalReferencesRange.Minimum,
168 ↪ currentInternalReferenceIndex);
169     ExternalReferencesRange = possibleExternalReferencesRange;
170 }
171
172 /// <summary>
173 /// <para>
174 /// Initializes a new <see cref="LinksConstants"/> instance.
175 /// </para>
176 /// <para></para>
177 /// </summary>
178 /// <param name="targetPart">

```

```

174    /// <para>A target part.</para>
175    /// <para></para>
176    /// </param>
177    /// <param name="enableExternalReferencesSupport">
178    /// <para>A enable external references support.</para>
179    /// <para></para>
180    /// </param>
181    [MethodImpl(MethodImplOptions.AggressiveInlining)]
182    public LinksConstants(int targetPart, bool enableExternalReferencesSupport) :
        ↪ this(targetPart, GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
        ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }

183
184    /// <summary>
185    /// <para>
186    /// Initializes a new <see cref="LinksConstants"/> instance.
187    /// </para>
188    /// <para></para>
189    /// </summary>
190    /// <param name="possibleInternalReferencesRange">
191    /// <para>A possible internal references range.</para>
192    /// <para></para>
193    /// </param>
194    /// <param name="possibleExternalReferencesRange">
195    /// <para>A possible external references range.</para>
196    /// <para></para>
197    /// </param>
198    [MethodImpl(MethodImplOptions.AggressiveInlining)]
199    public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange,
        ↪ Range<TLinkAddress>? possibleExternalReferencesRange) : this(DefaultTargetPart,
        ↪ possibleInternalReferencesRange, possibleExternalReferencesRange) { }

200
201    /// <summary>
202    /// <para>
203    /// Initializes a new <see cref="LinksConstants"/> instance.
204    /// </para>
205    /// <para></para>
206    /// </summary>
207    /// <param name="enableExternalReferencesSupport">
208    /// <para>A enable external references support.</para>
209    /// <para></para>
210    /// </param>
211    [MethodImpl(MethodImplOptions.AggressiveInlining)]
212    public LinksConstants(bool enableExternalReferencesSupport) :
        ↪ this(GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
        ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }

213
214    /// <summary>
215    /// <para>
216    /// Initializes a new <see cref="LinksConstants"/> instance.
217    /// </para>
218    /// <para></para>
219    /// </summary>
220    /// <param name="targetPart">
221    /// <para>A target part.</para>
222    /// <para></para>
223    /// </param>
224    /// <param name="possibleInternalReferencesRange">
225    /// <para>A possible internal references range.</para>
226    /// <para></para>
227    /// </param>
228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
229    public LinksConstants(int targetPart, Range<TLinkAddress>
        ↪ possibleInternalReferencesRange) : this(targetPart, possibleInternalReferencesRange,
        ↪ null) { }

230
231    /// <summary>
232    /// <para>
233    /// Initializes a new <see cref="LinksConstants"/> instance.
234    /// </para>
235    /// <para></para>
236    /// </summary>
237    /// <param name="possibleInternalReferencesRange">
238    /// <para>A possible internal references range.</para>
239    /// <para></para>
240    /// </param>
241    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

242 public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange) :
    ↪ this(DefaultTargetPart, possibleInternalReferencesRange, null) { }
243
244 /// <summary>
245 /// <para>
246 /// Initializes a new <see cref="LinksConstants"/> instance.
247 /// </para>
248 /// <para></para>
249 /// </summary>
250 [MethodImpl(MethodImplOptions.AggressiveInlining)]
251 public LinksConstants() : this(DefaultTargetPart, enableExternalReferencesSupport:
    ↪ false) { }
252
253 /// <summary>
254 /// <para>
255 /// Gets the default internal references range using the specified enable external
    ↪ references support.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="enableExternalReferencesSupport">
260 /// <para>The enable external references support.</para>
261 /// <para></para>
262 /// </param>
263 /// <returns>
264 /// <para>A range of t link address</para>
265 /// <para></para>
266 /// </returns>
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 public static Range<TLinkAddress> GetDefaultInternalReferencesRange(bool
    ↪ enableExternalReferencesSupport)
269 {
270     if (enableExternalReferencesSupport)
271     {
272         return (_one, _uint64ToAddressConverter.Convert(Hybrid<TLinkAddress>.HalfOfNumbe
            ↪ rValuesRange));
273     }
274     else
275     {
276         return (_one, NumericType<TLinkAddress>.MaxValue);
277     }
278 }
279
280 /// <summary>
281 /// <para>
282 /// Gets the default external references range using the specified enable external
    ↪ references support.
283 /// </para>
284 /// <para></para>
285 /// </summary>
286 /// <param name="enableExternalReferencesSupport">
287 /// <para>The enable external references support.</para>
288 /// <para></para>
289 /// </param>
290 /// <returns>
291 /// <para>A range of t link address</para>
292 /// <para></para>
293 /// </returns>
294 [MethodImpl(MethodImplOptions.AggressiveInlining)]
295 public static Range<TLinkAddress>? GetDefaultExternalReferencesRange(bool
    ↪ enableExternalReferencesSupport)
296 {
297     if (enableExternalReferencesSupport)
298     {
299         return (Hybrid<TLinkAddress>.ExternalZero, NumericType<TLinkAddress>.MaxValue);
300     }
301     else
302     {
303         return null;
304     }
305 }
306 }
307 }

```

## 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>
46        /// <para></para>
47        /// </summary>
48        /// <typeparam name="TLinkAddress">
49        /// <para>The link address.</para>
50        /// <para></para>
51        /// </typeparam>
52        /// <param name="linksConstants">
53        /// <para>The links constants.</para>
54        /// <para></para>
55        /// </param>
56        /// <param name="address">
57        /// <para>The address.</para>
58        /// <para></para>
59        /// </param>
60        /// <returns>
61        /// <para>The bool</para>
62        /// <para></para>
63        /// </returns>
64    }
65 }

```

```

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         /// <summary>
24         /// <para>
25         /// The default.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
30             ↪ EqualityComparer<TLinkAddress>.Default;
31
32         /// <summary>
33         /// <para>
34         /// Gets the index value.
35         /// </para>

```

```

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

```

```

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

```

```

191 public IEnumerator<TLinkAddress> GetEnumerator()
192 {
193     for (int i = 0; i < Size; i++)
194     {
195         yield return Index;
196     }
197 }
198
199 /// <summary>
200 /// <para>
201 /// Indexes the of using the specified item.
202 /// </para>
203 /// <para></para>
204 /// </summary>
205 /// <param name="item">
206 /// <para>The item.</para>
207 /// <para></para>
208 /// </param>
209 /// <returns>
210 /// <para>The int</para>
211 /// <para></para>
212 /// </returns>
213 [MethodImpl(MethodImplOptions.AggressiveInlining)]
214 public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
    ↪ 0 : -1;
215
216 /// <summary>
217 /// <para>
218 /// Inserts the index.
219 /// </para>
220 /// <para></para>
221 /// </summary>
222 /// <param name="index">
223 /// <para>The index.</para>
224 /// <para></para>
225 /// </param>
226 /// <param name="item">
227 /// <para>The item.</para>
228 /// <para></para>
229 /// </param>
230 [MethodImpl(MethodImplOptions.AggressiveInlining)]
231 public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
232
233 /// <summary>
234 /// <para>
235 /// Determines whether this instance remove.
236 /// </para>
237 /// <para></para>
238 /// </summary>
239 /// <param name="item">
240 /// <para>The item.</para>
241 /// <para></para>
242 /// </param>
243 /// <returns>
244 /// <para>The bool</para>
245 /// <para></para>
246 /// </returns>
247 [MethodImpl(MethodImplOptions.AggressiveInlining)]
248 public bool Remove(TLinkAddress item) => throw new NotSupportedException();
249
250 /// <summary>
251 /// <para>
252 /// Removes the at using the specified index.
253 /// </para>
254 /// <para></para>
255 /// </summary>
256 /// <param name="index">
257 /// <para>The index.</para>
258 /// <para></para>
259 /// </param>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 public void RemoveAt(int index) => throw new NotSupportedException();
262
263 /// <summary>
264 /// <para>
265 /// Gets the enumerator.
266 /// </para>
267 /// <para></para>

```

```

268     /// </summary>
269     /// <returns>
270     /// <para>The enumerator</para>
271     /// <para></para>
272     /// </returns>
273     [MethodImpl(MethodImplOptions.AggressiveInlining)]
274     IEnumerator IEnumerable.GetEnumerator()
275     {
276         for (int i = 0; i < Size; i++)
277         {
278             yield return Index;
279         }
280     }
281
282     /// <summary>
283     /// <para>
284     /// Determines whether this instance equals.
285     /// </para>
286     /// <para></para>
287     /// </summary>
288     /// <param name="other">
289     /// <para>The other.</para>
290     /// <para></para>
291     /// </param>
292     /// <returns>
293     /// <para>The bool</para>
294     /// <para></para>
295     /// </returns>
296     [MethodImpl(MethodImplOptions.AggressiveInlining)]
297     public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
        ↪ _equalityComparer.Equals(Index, other.Index);
298
299     [MethodImpl(MethodImplOptions.AggressiveInlining)]
300     public static implicit operator TLinkAddress(Point<TLinkAddress> linkAddress) =>
        ↪ linkAddress.Index;
301
302     /// <summary>
303     /// <para>
304     /// Determines whether this instance equals.
305     /// </para>
306     /// <para></para>
307     /// </summary>
308     /// <param name="obj">
309     /// <para>The obj.</para>
310     /// <para></para>
311     /// </param>
312     /// <returns>
313     /// <para>The bool</para>
314     /// <para></para>
315     /// </returns>
316     [MethodImpl(MethodImplOptions.AggressiveInlining)]
317     public override bool Equals(object obj) => obj is Point<TLinkAddress> linkAddress ?
        ↪ Equals(linkAddress) : false;
318
319     /// <summary>
320     /// <para>
321     /// Gets the hash code.
322     /// </para>
323     /// <para></para>
324     /// </summary>
325     /// <returns>
326     /// <para>The int</para>
327     /// <para></para>
328     /// </returns>
329     [MethodImpl(MethodImplOptions.AggressiveInlining)]
330     public override int GetHashCode() => Index.GetHashCode();
331
332     /// <summary>
333     /// <para>
334     /// Returns the string.
335     /// </para>
336     /// <para></para>
337     /// </summary>
338     /// <returns>
339     /// <para>The string</para>
340     /// <para></para>
341     /// </returns>
342     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

343 public override string ToString() => Index.ToString();
344
345 [MethodImpl(MethodImplOptions.AggressiveInlining)]
346 public static bool operator ==(Point<TLinkAddress> left, Point<TLinkAddress> right)
347 {
348     if (left == null && right == null)
349     {
350         return true;
351     }
352     if (left == null)
353     {
354         return false;
355     }
356     return left.Equals(right);
357 }
358
359 [MethodImpl(MethodImplOptions.AggressiveInlining)]
360 public static bool operator !=(Point<TLinkAddress> left, Point<TLinkAddress> right) =>
361     ↪ !(left == right);
362
363 /// <summary>
364 /// <para>
365 /// Determines whether is full point.
366 /// </para>
367 /// <para></para>
368 /// </summary>
369 /// <param name="link">
370 /// <para>The link.</para>
371 /// <para></para>
372 /// </param>
373 /// <returns>
374 /// <para>The bool</para>
375 /// <para></para>
376 /// </returns>
377 [MethodImpl(MethodImplOptions.AggressiveInlining)]
378 public static bool IsFullPoint(params TLinkAddress[] link) =>
379     ↪ IsFullPoint((IList<TLinkAddress>)link);
380
381 /// <summary>
382 /// <para>
383 /// Determines whether is full point.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 /// <param name="link">
388 /// <para>The link.</para>
389 /// <para></para>
390 /// </param>
391 /// <returns>
392 /// <para>The bool</para>
393 /// <para></para>
394 /// </returns>
395 [MethodImpl(MethodImplOptions.AggressiveInlining)]
396 public static bool IsFullPoint(IList<TLinkAddress> link)
397 {
398     Ensure.Always.ArgumentNotEmpty(link, nameof(link));
399     Ensure.Always.ArgumentInRange(link.Count, (2, int.MaxValue), nameof(link), "Cannot
400     ↪ determine link's pointness using only its identifier.");
401     return IsFullPointUnchecked(link);
402 }
403
404 /// <summary>
405 /// <para>
406 /// Determines whether is full point unchecked.
407 /// </para>
408 /// <para></para>
409 /// </summary>
410 /// <param name="link">
411 /// <para>The link.</para>
412 /// <para></para>
413 /// </param>
414 /// <returns>
415 /// <para>The result.</para>
416 /// <para></para>
417 /// </returns>
418 [MethodImpl(MethodImplOptions.AggressiveInlining)]
419 public static bool IsFullPointUnchecked(IList<TLinkAddress> link)
420 {

```

```

418     var result = true;
419     for (var i = 1; result && i < link.Count; i++)
420     {
421         result = _equalityComparer.Equals(link[0], link[i]);
422     }
423     return result;
424 }
425
426 /// <summary>
427 /// <para>
428 /// Determines whether is partial point.
429 /// </para>
430 /// <para></para>
431 /// </summary>
432 /// <param name="link">
433 /// <para>The link.</para>
434 /// <para></para>
435 /// </param>
436 /// <returns>
437 /// <para>The bool</para>
438 /// <para></para>
439 /// </returns>
440 [MethodImpl(MethodImplOptions.AggressiveInlining)]
441 public static bool IsPartialPoint(params TLinkAddress[] link) =>
442     ↪ IsPartialPoint((IList<TLinkAddress>)link);
443
444 /// <summary>
445 /// <para>
446 /// Determines whether is partial point.
447 /// </para>
448 /// <para></para>
449 /// </summary>
450 /// <param name="link">
451 /// <para>The link.</para>
452 /// <para></para>
453 /// </param>
454 /// <returns>
455 /// <para>The bool</para>
456 /// <para></para>
457 /// </returns>
458 [MethodImpl(MethodImplOptions.AggressiveInlining)]
459 public static bool IsPartialPoint(IList<TLinkAddress> link)
460 {
461     Ensure.Always.ArgumentNotEmpty(link, nameof(link));
462     Ensure.Always.ArgumentInRange(link.Count, (2, int.MaxValue), nameof(link), "Cannot
463     ↪ determine link's pointness using only its identifier.");
464     return IsPartialPointUnchecked(link);
465 }
466
467 /// <summary>
468 /// <para>
469 /// Determines whether is partial point unchecked.
470 /// </para>
471 /// <para></para>
472 /// </summary>
473 /// <param name="link">
474 /// <para>The link.</para>
475 /// <para></para>
476 /// </param>
477 /// <returns>
478 /// <para>The result.</para>
479 /// <para></para>
480 /// </returns>
481 [MethodImpl(MethodImplOptions.AggressiveInlining)]
482 public static bool IsPartialPointUnchecked(IList<TLinkAddress> link)
483 {
484     var result = false;
485     for (var i = 1; !result && i < link.Count; i++)
486     {
487         result = _equalityComparer.Equals(link[0], link[i]);
488     }
489     return result;
490 }

```

## 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>
74         /// <returns>
75         /// <para>The link address</para>
76         /// <para></para>
77         /// </returns>

```

```

74     TLinkAddress Trigger(IList<TLinkAddress> restriction, Func<IList<TLinkAddress>,
75         ↳ IList<TLinkAddress>, TLinkAddress> matchedHandler,
76         IList<TLinkAddress> substitution, Func<IList<TLinkAddress>, IList<TLinkAddress>,
77         ↳ TLinkAddress> substitutedHandler);
78 }
79
80 /// <remarks>Extended with small optimization.</remarks>
81 public partial interface IUniLinks<TLinkAddress>
82 {
83     /// <remarks>
84     /// Something simple should be simple and optimized.
85     /// </remarks>
86     TLinkAddress Count(IList<TLinkAddress> restrictions);
87 }
88 }

```

## 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 &amp; ...
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 &amp;&amp; after != null
40         /// * default(TLinkAddress) if before == null &amp;&amp; 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     }
32 }

```

## 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 }

```



## 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         /// <summary>
45         /// <para>
46         /// Tests the external references.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <typeparam name="TUnsigned">
51         /// <para>The unsigned.</para>
52         /// <para></para>
53         /// </typeparam>
54         /// <typeparam name="TSigned">
55         /// <para>The signed.</para>
56         /// <para></para>
57         /// </typeparam>
58         private static void TestExternalReferences<TUnsigned, TSigned>()
59         {
60             var unsingedOne = Arithmetic.Increment(default(TUnsigned));
61             var converter = UncheckedConverter<TSigned, TUnsigned>.Default;
62             var half = converter.Convert(NumericType<TSigned>.MaxValue);
63             LinksConstants<TUnsigned> constants = new LinksConstants<TUnsigned>((unsingedOne,
64                 ↪ half), (Arithmetic.Add(half, unsingedOne), NumericType<TUnsigned>.MaxValue));
65
66             var minimum = new Hybrid<TUnsigned>(default, isExternal: true);
67             var maximum = new Hybrid<TUnsigned>(half, isExternal: true);
68
69             Assert.True(constants.IsExternalReference(minimum));
70             Assert.True(minimum.IsExternal);
71             Assert.False(minimum.IsInternal);
72             Assert.True(constants.IsExternalReference(maximum));
73             Assert.True(maximum.IsExternal);
74             Assert.False(maximum.IsInternal);
75         }
76     }

```

76 }  
77 }

## Index

- ./csharp/Platform.Data.Tests/HybridTests.cs, 40
- ./csharp/Platform.Data.Tests/LinksConstantsTests.cs, 40
- ./csharp/Platform.Data/Exceptions/ArgumentLinkDoesNotExistException.cs, 1
- ./csharp/Platform.Data/Exceptions/ArgumentLinkHasDependenciesException.cs, 2
- ./csharp/Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsException.cs, 4
- ./csharp/Platform.Data/Exceptions/LinksLimitReachedException.cs, 5
- ./csharp/Platform.Data/Exceptions/LinksLimitReachedExceptionBase.cs, 6
- ./csharp/Platform.Data/Hybrid.cs, 7
- ./csharp/Platform.Data/ILinks.cs, 12
- ./csharp/Platform.Data/ILinksExtensions.cs, 14
- ./csharp/Platform.Data/ISynchronizedLinks.cs, 16
- ./csharp/Platform.Data/LinkAddress.cs, 17
- ./csharp/Platform.Data/LinksConstants.cs, 21
- ./csharp/Platform.Data/LinksConstantsBase.cs, 25
- ./csharp/Platform.Data/LinksConstantsExtensions.cs, 26
- ./csharp/Platform.Data/Numbers/Raw/AddressToRawNumberConverter.cs, 27
- ./csharp/Platform.Data/Numbers/Raw/RawNumberToAddressConverter.cs, 27
- ./csharp/Platform.Data/Point.cs, 28
- ./csharp/Platform.Data/Universal/IUniLinks.cs, 34
- ./csharp/Platform.Data/Universal/IUniLinksCRUD.cs, 36
- ./csharp/Platform.Data/Universal/IUniLinksGS.cs, 37
- ./csharp/Platform.Data/Universal/IUniLinksIO.cs, 38
- ./csharp/Platform.Data/Universal/IUniLinksIOWithExtensions.cs, 39
- ./csharp/Platform.Data/Universal/IUniLinksRW.cs, 39