

# 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{TLinkAddress}"/>
20         ///     instance.
21         /// </para>
22         /// <para>
23         /// Инициализирует новый экземпляр класса <see
24         ///     cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         /// <param name="link">
29         /// <para>A link.</para>
30         /// <para>Связь.</para>
31         /// </param>
32         /// <param name="argumentName">
33         /// <para>A argument name.</para>
34         /// <para>Имя аргумента.</para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public ArgumentLinkDoesNotExistsException(TLinkAddress link, string argumentName) :
38             base(FormatMessage(link, argumentName), argumentName) { }
39
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>
43         ///     instance.
44         /// </para>
45         /// <para>
46         /// Инициализирует новый экземпляр класса <see
47         ///     cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="link">
52         /// <para>A link.</para>
53         /// <para>Связь.</para>
54         /// </param>
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         public ArgumentLinkDoesNotExistsException(TLinkAddress link) : base(FormatMessage(link))
57         { }
58
59         /// <summary>
60         /// <para>
61         /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>
62         ///     instance.
63         /// </para>
64         /// <para>
65         /// Инициализирует новый экземпляр класса <see
66         ///     cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>.
67         /// </para>
68         /// </summary>
69         /// <param name="message">
70         /// <para>A message.</para>
71         /// <para>Сообщение.</para>
72         /// </param>
73         /// <param name="innerException">
74         /// <para>A inner exception.</para>
75         /// <para>Внутренняя ошибка.</para>
76         /// </param>
```

```

68     /// </param>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     public ArgumentLinkDoesNotExistsException(string message, Exception innerException) :
71         ↪ base(message, innerException) { }
72
73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>
76     ↪ instance.
77     /// </para>
78     /// <para>
79     /// Инициализирует новый экземпляр класса <see
80     ↪ cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>.
81     /// </para>
82     /// </summary>
83     /// <param name="message">
84     /// <para>A message.</para>
85     /// <para>Сообщение.</para>
86     /// </param>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     public ArgumentLinkDoesNotExistsException(string message) : base(message) { }
89
90     /// <summary>
91     /// <para>
92     /// Initializes a new <see cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>
93     ↪ instance.
94     /// </para>
95     /// <para>
96     /// Инициализирует новый экземпляр класса <see
97     ↪ cref="ArgumentLinkDoesNotExistsException{TLinkAddress}"/>.
98     /// </para>
99     /// </summary>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    public ArgumentLinkDoesNotExistsException() { }
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    private static string FormatMessage(TLinkAddress link, string argumentName) => $"Связь
104    ↪ [{link}] переданная в аргумент [{argumentName}] не существует.";
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    private static string FormatMessage(TLinkAddress link) => $"Связь [{link}] переданная в
107    ↪ качестве аргумента не существует.";
108 }
109 }

```

## 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{TLinkAddress}"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="link">
25         /// <para>A link.</para>
26         /// <para>Связь.</para>
27         /// </param>
28         /// <param name="paramName">
29         /// <para>A param name.</para>
30         /// <para>Имя параметра.</para>
31         /// </param>
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public ArgumentLinkHasDependenciesException(TLinkAddress link, string paramName) :
34             ↪ base(FormatMessage(link, paramName), paramName) { }

```

```

33
34     /// <summary>
35     /// <para>
36     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException{TLinkAddress}"/>
37     → 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{TLinkAddress}"/>
52     → instance.
53     /// </para>
54     /// <para></para>
55     /// </summary>
56     /// <param name="message">
57     /// <para>A message.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="innerException">
61     /// <para>A inner exception.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public ArgumentLinkHasDependenciesException(string message, Exception innerException) :
66     → base(message, innerException) { }
67
68     /// <summary>
69     /// <para>
70     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException{TLinkAddress}"/>
71     → instance.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <param name="message">
76     /// <para>A message.</para>
77     /// <para></para>
78     /// </param>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public ArgumentLinkHasDependenciesException(string message) : base(message) { }
81
82     /// <summary>
83     /// <para>
84     /// Initializes a new <see cref="ArgumentLinkHasDependenciesException{TLinkAddress}"/>
85     → instance.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public ArgumentLinkHasDependenciesException() { }
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     private static string FormatMessage(TLinkAddress link, string paramName) => $"У связи
93     → [{link}] переданной в аргумент [{paramName}] присутствуют зависимости, которые
94     → препятствуют изменению её внутренней структуры.";
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     private static string FormatMessage(TLinkAddress link) => $"У связи [{link}] переданной
97     → в качестве аргумента присутствуют зависимости, которые препятствуют изменению её
98     → внутренней структуры.";
99 }
100 }

```

### 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
24             ↳ существует.";
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="message">
33         /// <para>A message.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="innerException">
37         /// <para>A inner exception.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public LinkWithSameValueAlreadyExistsException(string message, Exception innerException)
42             ↳ : base(message, innerException) { }
43
44         /// <summary>
45         /// <para>
46         /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="message">
51         /// <para>A message.</para>
52         /// <para></para>
53         /// </param>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public LinkWithSameValueAlreadyExistsException(string message) : base(message) { }
56
57         /// <summary>
58         /// <para>
59         /// Initializes a new <see cref="LinkWithSameValueAlreadyExistsException"/> instance.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         public LinkWithSameValueAlreadyExistsException() : base(DefaultMessage) { }
65     }
66 }

```

#### 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{TLinkAddress}"/> 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{TLinkAddress}" /> 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{TLinkAddress}" /> instance.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="message">
55     /// <para>A message.</para>
56     /// <para></para>
57     /// </param>
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public LinksLimitReachedException(string message) : base(message) { }
60
61     /// <summary>
62     /// <para>
63     /// Initializes a new <see cref="LinksLimitReachedException{TLinkAddress}" /> instance.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public LinksLimitReachedException() : base(DefaultMessage) { }
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     private static string FormatMessage(TLinkAddress limit) => $"Достигнут лимит количества
71     ↪ связей в хранилище ({limit}).";
72 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Exceptions
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links limit reached exception base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="Exception"/>
15     public abstract class LinksLimitReachedExceptionBase : Exception
16     {
17         /// <summary>
18         /// <para>
19         /// The default message.
20         /// </para>
21         /// <para></para>
22         /// </summary>

```

```

23     public static readonly string DefaultMessage = "Достигнут лимит количества связей в
    ↳ хранилище.";
24
25     /// <summary>
26     /// <para>
27     /// Initializes a new <see cref="LinksLimitReachedExceptionBase"/> instance.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     /// <param name="message">
32     /// <para>A message.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="innerException">
36     /// <para>A inner exception.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     protected LinksLimitReachedExceptionBase(string message, Exception innerException) :
    ↳ base(message, innerException) { }
41
42     /// <summary>
43     /// <para>
44     /// Initializes a new <see cref="LinksLimitReachedExceptionBase"/> instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="message">
49     /// <para>A message.</para>
50     /// <para></para>
51     /// </param>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected LinksLimitReachedExceptionBase(string message) : base(message) { }
54 }
55 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Exceptions;
5  using Platform.Reflection;
6  using Platform.Converters;
7  using Platform.Numbers;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data
12 {
13     /// <summary>
14     /// <para>
15     /// The hybrid.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public struct Hybrid<TLinkAddress> : IEquatable<Hybrid<TLinkAddress>>
20     {
21         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
    ↳ EqualityComparer<TLinkAddress>.Default;
22         private static readonly UncheckedSignExtendingConverter<TLinkAddress, long>
    ↳ _addressToInt64Converter = UncheckedSignExtendingConverter<TLinkAddress,
    ↳ long>.Default;
23         private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
    ↳ = UncheckedConverter<long, TLinkAddress>.Default;
24         private static readonly UncheckedConverter<TLinkAddress, ulong>
    ↳ _addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
25         private static readonly UncheckedConverter<ulong, TLinkAddress>
    ↳ _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
26         private static readonly UncheckedConverter<object, long> _objectToInt64Converter =
    ↳ UncheckedConverter<object, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The max value.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         public static readonly ulong HalfOfNumberValuesRange =
    ↳ _addressToUInt64Converter.Convert(NumericType<TLinkAddress>.MaxValue) / 2;

```

```

35     /// <summary>
36     /// <para>
37     /// The half of number values range.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public static readonly TLinkAddress ExternalZero =
42         ↪ _uInt64ToAddressConverter.Convert(HalfOfNumberValuesRange + 1UL);
43
44     /// <summary>
45     /// <para>
46     /// The value.
47     /// </para>
48     /// <para></para>
49     /// </summary>
50     public readonly TLinkAddress Value;
51
52     /// <summary>
53     /// <para>
54     /// Gets the is nothing value.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     public bool IsNothing
59     {
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue == 0;
62     }
63
64     /// <summary>
65     /// <para>
66     /// Gets the is internal value.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     public bool IsInternal
71     {
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         get => SignedValue > 0;
74     }
75
76     /// <summary>
77     /// <para>
78     /// Gets the is external value.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     public bool IsExternal
83     {
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]
85         get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue < 0;
86     }
87
88     /// <summary>
89     /// <para>
90     /// Gets the signed value value.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     public long SignedValue
95     {
96         [MethodImpl(MethodImplOptions.AggressiveInlining)]
97         get => _addressToInt64Converter.Convert(Value);
98     }
99
100     /// <summary>
101     /// <para>
102     /// Gets the absolute value value.
103     /// </para>
104     /// <para></para>
105     /// </summary>
106     public long AbsoluteValue
107     {
108         [MethodImpl(MethodImplOptions.AggressiveInlining)]
109         get => _equalityComparer.Equals(Value, ExternalZero) ? 0 :
110             ↪ Platform.Numbers.Math.Abs(SignedValue);
111     }
112
113     /// <summary>

```

```

112    /// <para>
113    /// Initializes a new <see cref="Hybrid{TLinkAddress}"/> instance.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="value">
118    /// <para>A value.</para>
119    /// <para></para>
120    /// </param>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    public Hybrid(TLinkAddress value)
123    {
124        Ensure.OnDebug.IsUnsignedInteger<TLinkAddress>();
125        Value = value;
126    }
127
128    /// <summary>
129    /// <para>
130    /// Initializes a new <see cref="Hybrid{TLinkAddress}"/> instance.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="value">
135    /// <para>A value.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="isExternal">
139    /// <para>A is external.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    public Hybrid(TLinkAddress value, bool isExternal)
144    {
145        if (_equalityComparer.Equals(value, default) && isExternal)
146        {
147            Value = ExternalZero;
148        }
149        else
150        {
151            if (isExternal)
152            {
153                Value = Math<TLinkAddress>.Negate(value);
154            }
155            else
156            {
157                Value = value;
158            }
159        }
160    }
161
162    /// <summary>
163    /// <para>
164    /// Initializes a new <see cref="Hybrid{TLinkAddress}"/> instance.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="value">
169    /// <para>A value.</para>
170    /// <para></para>
171    /// </param>
172    [MethodImpl(MethodImplOptions.AggressiveInlining)]
173    public Hybrid(object value) => Value =
174        ↪ _int64ToAddressConverter.Convert(_objectToInt64Converter.Convert(value));
175
176    /// <summary>
177    /// <para>
178    /// Initializes a new <see cref="Hybrid{TLinkAddress}"/> instance.
179    /// </para>
180    /// <para></para>
181    /// </summary>
182    /// <param name="value">
183    /// <para>A value.</para>
184    /// <para></para>
185    /// </param>
186    /// <param name="isExternal">
187    /// <para>A is external.</para>
188    /// <para></para>
189    /// </param>

```



```

189 [MethodImpl(MethodImplOptions.AggressiveInlining)]
190 public Hybrid(object value, bool isExternal)
191 {
192     var signedValue = value == null ? 0 : _objectToInt64Converter.Convert(value);
193     if (signedValue == 0 && isExternal)
194     {
195         Value = ExternalZero;
196     }
197     else
198     {
199         var absoluteValue = System.Math.Abs(signedValue);
200         Value = isExternal ? _int64ToAddressConverter.Convert(-absoluteValue) :
            ↪ _int64ToAddressConverter.Convert(absoluteValue);
201     }
202 }
203
204 [MethodImpl(MethodImplOptions.AggressiveInlining)]
205 public static implicit operator Hybrid<TLinkAddress>(TLinkAddress integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
206
207 [MethodImpl(MethodImplOptions.AggressiveInlining)]
208 public static explicit operator Hybrid<TLinkAddress>(ulong integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
209
210 [MethodImpl(MethodImplOptions.AggressiveInlining)]
211 public static explicit operator Hybrid<TLinkAddress>(long integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
212
213 [MethodImpl(MethodImplOptions.AggressiveInlining)]
214 public static explicit operator Hybrid<TLinkAddress>(uint integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
215
216 [MethodImpl(MethodImplOptions.AggressiveInlining)]
217 public static explicit operator Hybrid<TLinkAddress>(int integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
218
219 [MethodImpl(MethodImplOptions.AggressiveInlining)]
220 public static explicit operator Hybrid<TLinkAddress>(ushort integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
221
222 [MethodImpl(MethodImplOptions.AggressiveInlining)]
223 public static explicit operator Hybrid<TLinkAddress>(short integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
224
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 public static explicit operator Hybrid<TLinkAddress>(byte integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
227
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 public static explicit operator Hybrid<TLinkAddress>(sbyte integer) => new
    ↪ Hybrid<TLinkAddress>(integer);
230
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 public static implicit operator TLinkAddress(Hybrid<TLinkAddress> hybrid) =>
    ↪ hybrid.Value;
233
234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 public static explicit operator ulong(Hybrid<TLinkAddress> hybrid) =>
    ↪ CheckedConverter<TLinkAddress, ulong>.Default.Convert(hybrid.Value);
236
237 [MethodImpl(MethodImplOptions.AggressiveInlining)]
238 public static explicit operator long(Hybrid<TLinkAddress> hybrid) =>
    ↪ hybrid.AbsoluteValue;
239
240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
241 public static explicit operator uint(Hybrid<TLinkAddress> hybrid) =>
    ↪ CheckedConverter<TLinkAddress, uint>.Default.Convert(hybrid.Value);
242
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 public static explicit operator int(Hybrid<TLinkAddress> hybrid) =>
    ↪ (int)hybrid.AbsoluteValue;
245
246 [MethodImpl(MethodImplOptions.AggressiveInlining)]
247 public static explicit operator ushort(Hybrid<TLinkAddress> hybrid) =>
    ↪ CheckedConverter<TLinkAddress, ushort>.Default.Convert(hybrid.Value);
248
249 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

250 public static explicit operator short(Hybrid<TLinkAddress> hybrid) =>
251     ↪ (short)hybrid.AbsoluteValue;
252
253 [MethodImpl(MethodImplOptions.AggressiveInlining)]
254 public static explicit operator byte(Hybrid<TLinkAddress> hybrid) =>
255     ↪ CheckedConverter<TLinkAddress, byte>.Default.Convert(hybrid.Value);
256
257 [MethodImpl(MethodImplOptions.AggressiveInlining)]
258 public static explicit operator sbyte(Hybrid<TLinkAddress> hybrid) =>
259     ↪ (sbyte)hybrid.AbsoluteValue;
260
261 /// <summary>
262 /// <para>
263 /// Returns the string.
264 /// </para>
265 /// <para></para>
266 /// </summary>
267 /// <returns>
268 /// <para>The string</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 public override string ToString() => IsExternal ? $"{<AbsoluteValue>}" :
273     ↪ Value.ToString();
274
275 /// <summary>
276 /// <para>
277 /// Determines whether this instance equals.
278 /// </para>
279 /// <para></para>
280 /// </summary>
281 /// <param name="other">
282 /// <para>The other.</para>
283 /// <para></para>
284 /// </param>
285 /// <returns>
286 /// <para>The bool</para>
287 /// <para></para>
288 /// </returns>
289 [MethodImpl(MethodImplOptions.AggressiveInlining)]
290 public bool Equals(Hybrid<TLinkAddress> other) => _equalityComparer.Equals(Value,
291     ↪ other.Value);
292
293 /// <summary>
294 /// <para>
295 /// Determines whether this instance equals.
296 /// </para>
297 /// <para></para>
298 /// </summary>
299 /// <param name="obj">
300 /// <para>The obj.</para>
301 /// <para></para>
302 /// </param>
303 /// <returns>
304 /// <para>The bool</para>
305 /// <para></para>
306 /// </returns>
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 public override bool Equals(object obj) => obj is Hybrid<TLinkAddress> hybrid ?
309     ↪ Equals(hybrid) : false;
310
311 /// <summary>
312 /// <para>
313 /// Gets the hash code.
314 /// </para>
315 /// <para></para>
316 /// </summary>
317 /// <returns>
318 /// <para>The int</para>
319 /// <para></para>
320 /// </returns>
321 [MethodImpl(MethodImplOptions.AggressiveInlining)]
322 public override int GetHashCode() => Value.GetHashCode();
323
324 [MethodImpl(MethodImplOptions.AggressiveInlining)]
325 public static bool operator ==(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
326     ↪ left.Equals(right);

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
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>Represents an interface for manipulating data in the Links (links storage)
12     ↪     format.</para>
13     /// <para>Представляет интерфейс для манипуляции с данными в формате Links (хранилища
14     ↪     связей).</para>
15     /// </summary>
16     /// <remarks>
17     /// <para>This interface is independent of the size of the content of the link, meaning it
18     ↪     is suitable for both doublets, triplets, and link sequences of any size.</para>
19     /// <para>Этот интерфейс не зависит от размера содержимого связи, а значит подходит как для
20     ↪     дуплетов, триплетов и последовательностей связей любого размера.</para>
21     /// </remarks>
22     public interface ILinks<TLinkAddress, TConstants>
23     where TConstants : LinksConstants<TLinkAddress>
24     {
25         #region Constants
26
27         /// <summary>
28         /// <para>Returns the set of constants that is necessary for effective communication
29         ↪     with the methods of this interface.</para>
30         /// <para>Возвращает набор констант, который необходим для эффективной коммуникации с
31         ↪     методами этого интерфейса.</para>
32         /// </summary>
33         /// <remarks>
34         /// <para>These constants are not changed since the creation of the links storage access
35         ↪     point.</para>
36         /// <para>Эти константы не меняются с момента создания точки доступа к хранилищу
37         ↪     связей.</para>
38         /// </remarks>
39         TConstants Constants
40         {
41             [MethodImpl(MethodImplOptions.AggressiveInlining)]
42             get;
43         }
44
45         #endregion
46
47         #region Read
48
49         /// <summary>
50         /// <para>Counts and returns the total number of links in the storage that meet the
51         ↪     specified restriction.</para>
52         /// <para>Подсчитывает и возвращает общее число связей находящихся в хранилище,
53         ↪     соответствующих указанному ограничению.</para>
54         /// </summary>
55         /// <param name="restriction"><para>Restriction on the contents of
56         ↪     links.</para><para>Ограничение на содержимое связей.</para></param>
57         /// <returns><para>The total number of links in the storage that meet the specified
58         ↪     restriction.</para><para>Общее число связей находящихся в хранилище, соответствующих
59         ↪     указанному ограничению.</para></returns>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         TLinkAddress Count(IList<TLinkAddress> restriction);
62
63         /// <summary>
64         /// <para>Passes through all the links matching the pattern, invoking a handler for each
65         ↪     matching link.</para>
66         /// <para>Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
67         ↪     (handler) для каждой подходящей связи.</para>
68         /// </summary>
69         /// <param name="restriction">

```

```

55  /// <para>Restriction 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>Ограничение 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  /// <param name="handler"><para>A handler for each matching link.</para><para>Обработчик
    ↳ для каждой подходящей связи.</para></param>
59  /// <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>
60  [MethodImpl(MethodImplOptions.AggressiveInlining)]
61  TLinkAddress Each(IList<TLinkAddress> restriction, ReadHandler<TLinkAddress> handler);
62
63  #endregion
64
65  #region Write
66
67  /// <summary>
68  /// <para>Creates a link.</para>
69  /// <para>Создаёт связь.</para>
70  /// <param name="substitution">
71  /// <para>The content of a new link. This argument is optional, if the null passed as
    ↳ value that means no content of a link is set.</para>
72  /// <para>Содержимое новой связи. Этот аргумент опционален, если null передан в качестве
    ↳ значения это означает, что никакого содержимого для связи не установлено.</para>
73  /// </param>
74  /// <param name="handler">
75  /// <para>A function to handle each executed change. This function can use
    ↳ Constants.Continue to continue process each change. Constants.Break can be used to
    ↳ stop receiving of executed changes.</para>
76  /// <para>Функция для обработки каждого выполненного изменения. Эта функция может
    ↳ использовать Constants.Continue чтобы продолжить обрабатывать каждое изменение.
    ↳ Constants.Break может быть использована для остановки получения выполненных
    ↳ изменений.</para>
77  /// </param>
78  /// </summary>
79  /// <returns>
80  /// <para>
81  /// Constants.Continue if all executed changes are handled.
82  /// Constants.Break if processing of handled changes is stoped.
83  /// </para>
84  /// <para>
85  /// Constants.Continue если все выполненные изменения обработаны.
86  /// Constants.Break если обработка выполненных изменений остановлена.
87  /// </para>
88  /// </returns>
89  [MethodImpl(MethodImplOptions.AggressiveInlining)]
90  TLinkAddress Create(IList<TLinkAddress> substitution, WriteHandler<TLinkAddress>
    ↳ handler);
91
92  /// <summary>
93  /// Обновляет связь с указанными restriction[Constants.IndexPart] в адресом связи
94  /// на связь с указанным новым содержимым.
95  /// </summary>
96  /// <param name="restriction">
97  /// Ограничение на содержимое связей.
98  /// Предполагается, что будет указан индекс связи (в restriction[Constants.IndexPart]) и
    ↳ далее за ним будет следовать содержимое связи.
99  /// Каждое ограничение может иметь значения: Constants.Null - 0-я связь, обозначающая
    ↳ ссылку на пустоту,
100  /// Constants.Itself - требование установить ссылку на себя, 1.. $\infty$  конкретный индекс
    ↳ другой связи.
101  /// </param>
102  /// <param name="substitution"></param>
103  /// <param name="handler">
104  /// <para>A function to handle each executed change. This function can use
    ↳ Constants.Continue to continue process each change. Constants.Break can be used to
    ↳ stop receiving of executed changes.</para>
105  /// <para>Функция для обработки каждого выполненного изменения. Эта функция может
    ↳ использовать Constants.Continue чтобы продолжить обрабатывать каждое изменение.
    ↳ Constants.Break может быть использована для остановки получения выполненных
    ↳ изменений.</para>
106  /// </param>
107  /// </returns>

```

```

108     /// <para>
109     /// Constants.Continue if all executed changes are handled.
110     /// Constants.Break if proccessing of handled changes is stoped.
111     /// </para>
112     /// <para>
113     /// Constants.Continue если все выполненные изменения обработаны.
114     /// Constants.Break если обработка выполненных изменений остановлена.
115     /// </para>
116     /// </returns>
117     [MethodImpl(MethodImplOptions.AggressiveInlining)]
118     TLinkAddress Update(IList<TLinkAddress> restriction, IList<TLinkAddress> substitution,
119         → WriteHandler<TLinkAddress> handler);
120
121     /// <summary>
122     /// <para>Deletes links that match the specified restriction.</para>
123     /// <para>Удаляет связи соответствующие указанному ограничению.</para>
124     /// </summary>
125     /// <param name="restriction">
126     /// <para>Restriction on the content of a link. This argument is optional, if the null
127     → passed as value that means no restriction on the content of a link are set.</para>
128     /// <para>Ограничение на содержимое связи. Этот аргумент опционален, если null передан в
129     → качестве значения это означает, что никаких ограничений на содержимое связи не
130     → установлено.</para>
131     /// </param>
132     /// <param name="handler">
133     /// <para>A function to handle each executed change. This function can use
134     → Constants.Continue to continue process each change. Constants.Break can be used to
135     → stop receiving of executed changes.</para>
136     /// <para>Функция для обработки каждого выполненного изменения. Эта функция может
137     → использовать Constants.Continue чтобы продолжить обрабатывать каждое изменение.
138     → Constants.Break может быть использована для остановки получения выполненных
139     → изменений.</para>
140     /// </param>
141     /// <returns>
142     /// <para>
143     /// Constants.Continue if all executed changes are handled.
144     /// Constants.Break if proccessing of handled changes is stoped.
145     /// </para>
146     /// <para>
147     /// Constants.Continue если все выполненные изменения обработаны.
148     /// Constants.Break если обработка выполненных изменений остановлена.
149     /// </para>
150     /// </returns>
151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
152     TLinkAddress Delete(IList<TLinkAddress> restriction, WriteHandler<TLinkAddress> handler);
153
154     #endregion
155 }
156 }

```

## 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  using Platform.Delegates;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the links extensions.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     public static class ILinksExtensions
19     {
20         public static TLink Create<TLink>(this ILinks<TLink, LinksConstants<TLink>> links) =>
21             → links.Create(null);
22
23         public static TLink Create<TLink>(this ILinks<TLink, LinksConstants<TLink>> links,
24             → IList<TLink> substitution)
25         {
26             var constants = links.Constants;
27             Setter<TLink, TLink> setter = new Setter<TLink, TLink>(constants.Continue,
28                 → constants.Break, constants.Null);

```

```

26         links.Create(substitution, setter.SetFirstFromSecondListAndReturnTrue);
27         return setter.Result;
28     }
29
30     public static TLink Update<TLink>(this ILinks<TLink, LinksConstants<TLink>> links,
31     ↪ IList<TLink> restriction, IList<TLink> substitution)
32     {
33         var constants = links.Constants;
34         Setter<TLink, TLink> setter = new(constants.Continue, constants.Break,
35         ↪ constants.Null);
36         links.Update(restriction, substitution, setter.SetFirstFromSecondListAndReturnTrue);
37         return setter.Result;
38     }
39
40     public static TLink Delete<TLink>(this ILinks<TLink, LinksConstants<TLink>> links, TLink
41     ↪ linkToDelete) => Delete(links, (IList<TLink>)new LinkAddress<TLink>(linkToDelete));
42
43     public static TLink Delete<TLink>(this ILinks<TLink, LinksConstants<TLink>> links,
44     ↪ IList<TLink> restriction)
45     {
46         var constants = links.Constants;
47         Setter<TLink, TLink> setter = new Setter<TLink, TLink>(constants.Continue,
48         ↪ constants.Break, constants.Null);
49         links.Delete(restriction, setter.SetFirstFromFirstListAndReturnTrue);
50         return setter.Result;
51     }
52
53     /// <summary>
54     /// <para>
55     /// Counts the links.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <typeparam name="TLinkAddress">
60     /// <para>The link address.</para>
61     /// <para></para>
62     /// </typeparam>
63     /// <typeparam name="TConstants">
64     /// <para>The constants.</para>
65     /// <para></para>
66     /// </typeparam>
67     /// <param name="links">
68     /// <para>The links.</para>
69     /// <para></para>
70     /// </param>
71     /// <param name="restrictions">
72     /// <para>The restrictions.</para>
73     /// <para></para>
74     /// </param>
75     /// <returns>
76     /// <para>The link address</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public static TLinkAddress Count<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
81     ↪ TConstants> links, params TLinkAddress[] restrictions)
82     where TConstants : LinksConstants<TLinkAddress>
83     => links.Count(restrictions);
84
85     /// <summary>
86     /// Возвращает значение, определяющее существует ли связь с указанным индексом в
87     ↪ хранилище связей.
88     /// </summary>
89     /// <param name="links">Хранилище связей.</param>
90     /// <param name="link">Индекс проверяемой на существование связи.</param>
91     /// <returns>Значение, определяющее существует ли связь.</returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public static bool Exists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
94     ↪ TConstants> links, TLinkAddress link)
95     where TConstants : LinksConstants<TLinkAddress>
96     {
97         var constants = links.Constants;
98         return constants.IsExternalReference(link) || (constants.IsInternalReference(link)
99         ↪ && Comparer<TLinkAddress>.Default.Compare(links.Count(new
100         ↪ LinkAddress<TLinkAddress>(link)), default) > 0);
101     }
102
103     /// <param name="links">Хранилище связей.</param>

```

```

94     /// <param name="link">Индекс проверяемой на существование связи.</param>
95     /// <remarks>
96     /// TODO: May be move to EnsureExtensions or make it both there and here
97     /// </remarks>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
100     ↪ TConstants> links, TLinkAddress link)
101     where TConstants : LinksConstants<TLinkAddress>
102     {
103         if (!links.Exists(link))
104         {
105             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link);
106         }
107     }
108     /// <param name="links">Хранилище связей.</param>
109     /// <param name="link">Индекс проверяемой на существование связи.</param>
110     /// <param name="argumentName">Имя аргумента, в который передается индекс связи.</param>
111     [MethodImpl(MethodImplOptions.AggressiveInlining)]
112     public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
113     ↪ TConstants> links, TLinkAddress link, string argumentName)
114     where TConstants : LinksConstants<TLinkAddress>
115     {
116         if (!links.Exists(link))
117         {
118             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
119         }
120     }
121     /// <summary>
122     /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
123     ↪ (handler) для каждой подходящей связи.
124     /// </summary>
125     /// <param name="links">Хранилище связей.</param>
126     /// <param name="handler">Обработчик каждой подходящей связи.</param>
127     /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
128     ↪ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
129     ↪ Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
130     /// <returns>True, в случае если проход по связям не был прерван и False в обратном
131     ↪ случае.</returns>
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     public static TLinkAddress Each<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
134     ↪ TConstants> links, ReadHandler<TLinkAddress> handler, params TLinkAddress[]
135     ↪ restrictions)
136     where TConstants : LinksConstants<TLinkAddress>
137     => links.Each(restrictions, handler);
138     /// <summary>
139     /// Возвращает части-значения для связи с указанным индексом.
140     /// </summary>
141     /// <param name="links">Хранилище связей.</param>
142     /// <param name="link">Индекс связи.</param>
143     /// <returns>Уникальную связь.</returns>
144     [MethodImpl(MethodImplOptions.AggressiveInlining)]
145     public static IList<TLinkAddress> GetLink<TLinkAddress, TConstants>(this
146     ↪ ILinks<TLinkAddress, TConstants> links, TLinkAddress link)
147     where TConstants : LinksConstants<TLinkAddress>
148     {
149         var constants = links.Constants;
150         if (constants.IsExternalReference(link))
151         {
152             return new Point<TLinkAddress>(link, constants.TargetPart + 1);
153         }
154         var linkPartsSetter = new Setter<IList<TLinkAddress>,
155         ↪ TLinkAddress>(constants.Continue, constants.Break);
156         links.Each(linkPartsSetter.SetAndReturnTrue, link);
157         return linkPartsSetter.Result;
158     }
159     #region Points
160     /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
161     ↪ точкой полностью (связью замкнутой на себе дважды).</summary>
162     /// <param name="links">Хранилище связей.</param>
163     /// <param name="link">Индекс проверяемой связи.</param>
164     /// <returns>Значение, определяющее является ли связь точкой полностью.</returns>
165     /// <remarks>

```

```

160     /// Связь точка - это связь, у которой начало (Source) и конец (Target) есть сама эта
161     /// → связь.
162     /// Но что, если точка уже есть, а нужно создать пару с таким же значением? Должны ли
163     /// → точка и пара существовать одновременно?
164     /// Или в качестве решения для точек нужно использовать 0 в качестве начала и конца, а
165     /// → сортировать по индексу в массиве связей?
166     /// Какое тогда будет значение Source и Target у точки? 0 или её индекс?
167     /// Или точка должна быть одновременно точкой и парой, а также последовательностями из
168     /// → самой себя любого размера?
169     /// Как только есть ссылка на себя, появляется этот парадокс, причём достаточно даже
170     /// → одной ссылки на себя (частичной точки).
171     /// А что если не выбирать что является точкой, пара нулей (цикл через пустоту) или
172     /// самостоятельный цикл через себя? Что если предоставить все варианты использования
173     /// → связей?
174     /// Что если разрешить и нули, а так же частичные варианты?
175     ///
176     /// Что если точка, это только в том случае когда link.Source == link && link.Target == link, т.е. дважды ссылка на себя.
177     /// А пара это тогда, когда link.Source == link.Target && link.Source != link, т.е. ссылка не на себя а во вне.
178     ///
179     /// Тогда если у нас уже создана пара, но нам нужна точка, мы можем используя
180     /// → промежуточную связь,
181     /// например "DoubletOf" обозначить что является точно парой, а что точно точкой.
182     /// И наоборот этот же метод поможет, если уже существует точка, но нам нужна пара.
183     /// </remarks>
184     [MethodImpl(MethodImplOptions.AggressiveInlining)]
185     public static bool IsFullPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
186     → TConstants> links, TLinkAddress link)
187     where TConstants : LinksConstants<TLinkAddress>
188     {
189         if (links.Constants.IsExternalReference(link))
190         {
191             return true;
192         }
193         links.EnsureLinkExists(link);
194         return Point<TLinkAddress>.IsFullPoint(links.GetLink(link));
195     }
196
197     /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
198     /// → точкой частично (связью замкнутой на себе как минимум один раз).</summary>
199     /// <param name="links">Хранилище связей.</param>
200     /// <param name="link">Индекс проверяемой связи.</param>
201     /// <returns>Значение, определяющее является ли связь точкой частично.</returns>
202     /// <remarks>
203     /// Достаточно любой одной ссылки на себя.
204     /// Также в будущем можно будет проверять и всех родителей, чтобы проверить есть ли
205     /// → ссылки на себя (на эту связь).
206     /// </remarks>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     public static bool IsPartialPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
209     → TConstants> links, TLinkAddress link)
210     where TConstants : LinksConstants<TLinkAddress>
211     {
212         if (links.Constants.IsExternalReference(link))
213         {
214             return true;
215         }
216         links.EnsureLinkExists(link);
217         return Point<TLinkAddress>.IsPartialPoint(links.GetLink(link));
218     }
219
220     #endregion
221 }

```

## 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         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
22             ↳ EqualityComparer<TLinkAddress>.Default;
23
24         /// <summary>
25         /// <para>
26         /// Gets the index value.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         public TLinkAddress Index
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get;
34         }
35
36         /// <summary>
37         /// <para>
38         /// The not supported exception.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         public TLinkAddress this[int index]
43         {
44             [MethodImpl(MethodImplOptions.AggressiveInlining)]
45             get
46             {
47                 {
48                     if (index == 0)
49                     {
50                         return Index;
51                     }
52                     else
53                     {
54                         throw new IndexOutOfRangeException();
55                     }
56                 }
57             }
58             [MethodImpl(MethodImplOptions.AggressiveInlining)]
59             set => throw new NotSupportedException();
60         }
61
62         /// <summary>
63         /// <para>
64         /// Gets the count value.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         public int Count
69         {

```

```

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

```

```

144     /// <param name="arrayIndex">
145     /// <para>The array index.</para>
146     /// <para></para>
147     /// </param>
148     [MethodImpl(MethodImplOptions.AggressiveInlining)]
149     public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
150
151     /// <summary>
152     /// <para>
153     /// Gets the enumerator.
154     /// </para>
155     /// <para></para>
156     /// </summary>
157     /// <returns>
158     /// <para>An enumerator of t link address</para>
159     /// <para></para>
160     /// </returns>
161     [MethodImpl(MethodImplOptions.AggressiveInlining)]
162     public IEnumerator<TLinkAddress> GetEnumerator()
163     {
164         yield return Index;
165     }
166
167     /// <summary>
168     /// <para>
169     /// Indexes the of using the specified item.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="item">
174     /// <para>The item.</para>
175     /// <para></para>
176     /// </param>
177     /// <returns>
178     /// <para>The int</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
        0 : -1;
183
184     /// <summary>
185     /// <para>
186     /// Inserts the index.
187     /// </para>
188     /// <para></para>
189     /// </summary>
190     /// <param name="index">
191     /// <para>The index.</para>
192     /// <para></para>
193     /// </param>
194     /// <param name="item">
195     /// <para>The item.</para>
196     /// <para></para>
197     /// </param>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance remove.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="item">
208     /// <para>The item.</para>
209     /// <para></para>
210     /// </param>
211     /// <returns>
212     /// <para>The bool</para>
213     /// <para></para>
214     /// </returns>
215     [MethodImpl(MethodImplOptions.AggressiveInlining)]
216     public bool Remove(TLinkAddress item) => throw new NotSupportedException();
217
218     /// <summary>
219     /// <para>
220     /// Removes the at using the specified index.

```

```

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

```

```

295     /// <para></para>
296     /// </returns>
297     [MethodImpl(MethodImplOptions.AggressiveInlining)]
298     public override int GetHashCode() => Index.GetHashCode();
299
300     /// <summary>
301     /// <para>
302     /// Returns the string.
303     /// </para>
304     /// <para></para>
305     /// </summary>
306     /// <returns>
307     /// <para>The string</para>
308     /// <para></para>
309     /// </returns>
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     public override string ToString() => Index.ToString();
312
313     [MethodImpl(MethodImplOptions.AggressiveInlining)]
314     public static bool operator ==(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
    ↪ right)
315     {
316         if (left == null && right == null)
317         {
318             return true;
319         }
320         if (left == null)
321         {
322             return false;
323         }
324         return left.Equals(right);
325     }
326
327     [MethodImpl(MethodImplOptions.AggressiveInlining)]
328     public static bool operator !=(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
    ↪ right) => !(left == right);
329 }
330 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Ranges;
3  using Platform.Reflection;
4  using Platform.Converters;
5  using Platform.Numbers;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the links constants.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksConstantsBase"/>
18     public class LinksConstants<TLinkAddress> : LinksConstantsBase
19     {
20         private static readonly TLinkAddress _one = Arithmetic<TLinkAddress>.Increment(default);
21         private static readonly UncheckedConverter<ulong, TLinkAddress>
    ↪ _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
22
23         #region Link parts
24
25         /// <summary>Возвращает индекс части, которая отвечает за индекс (адрес, идентификатор)
    ↪ самой связи.</summary>
26         public int IndexPart
27         {
28             [MethodImpl(MethodImplOptions.AggressiveInlining)]
29             get;
30         }
31
32         /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-начало (первая
    ↪ часть-значение).</summary>
33         public int SourcePart
34         {
35             [MethodImpl(MethodImplOptions.AggressiveInlining)]
36             get;

```

```

37     }
38
39     /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-конец
    ↳ (последняя часть-значение).</summary>
40     public int TargetPart
41     {
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         get;
44     }
45
46     #endregion
47
48     #region Flow control
49
50     /// <summary>Возвращает значение, обозначающее продолжение прохода по связям.</summary>
51     /// <remarks>Используется в функции обработчике, который передаётся в функцию
    ↳ Each.</remarks>
52     public TLinkAddress Continue
53     {
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         get;
56     }
57
58     /// <summary>Возвращает значение, обозначающее пропуск в проходе по связям.</summary>
59     public TLinkAddress Skip
60     {
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         get;
63     }
64
65     /// <summary>Возвращает значение, обозначающее остановку прохода по связям.</summary>
66     /// <remarks>Используется в функции обработчике, который передаётся в функцию
    ↳ Each.</remarks>
67     public TLinkAddress Break
68     {
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         get;
71     }
72
73     #endregion
74
75     #region Special symbols
76
77     /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
78     public TLinkAddress Null
79     {
80         [MethodImpl(MethodImplOptions.AggressiveInlining)]
81         get;
82     }
83
84     /// <summary>Возвращает значение, обозначающее любую связь.</summary>
85     /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
    ↳ создавать все варианты последовательностей в функции Create.</remarks>
86     public TLinkAddress Any
87     {
88         [MethodImpl(MethodImplOptions.AggressiveInlining)]
89         get;
90     }
91
92     /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
93     public TLinkAddress Itself
94     {
95         [MethodImpl(MethodImplOptions.AggressiveInlining)]
96         get;
97     }
98
99     #endregion
100
101     #region References
102
103     /// <summary>Возвращает диапазон возможных индексов для внутренних связей (внутренних
    ↳ ссылок).</summary>
104     public Range<TLinkAddress> InternalReferencesRange
105     {
106         [MethodImpl(MethodImplOptions.AggressiveInlining)]
107         get;
108     }
109
110     /// <summary>Возвращает диапазон возможных индексов для внешних связей (внешних
    ↳ ссылок).</summary>

```

```

111 public Range<TLinkAddress>? ExternalReferencesRange
112 {
113     [MethodImpl(MethodImplOptions.AggressiveInlining)]
114     get;
115 }
116
117 #endregion
118
119 /// <summary>
120 /// <para>
121 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
122 /// </para>
123 /// </summary>
124 /// <param name="targetPart">
125 /// <para>A target part.</para>
126 /// </param>
127 /// <param name="possibleInternalReferencesRange">
128 /// <para>A possible internal references range.</para>
129 /// </param>
130 /// <param name="possibleExternalReferencesRange">
131 /// <para>A possible external references range.</para>
132 /// </param>
133 [MethodImpl(MethodImplOptions.AggressiveInlining)]
134 public LinksConstants(int targetPart, Range<TLinkAddress>
135     ↪ possibleInternalReferencesRange, Range<TLinkAddress>?
136     ↪ possibleExternalReferencesRange)
137 {
138     IndexPart = 0;
139     SourcePart = 1;
140     TargetPart = targetPart;
141     Null = default;
142     Break = default;
143     var currentInternalReferenceIndex = possibleInternalReferencesRange.Maximum;
144     Continue = currentInternalReferenceIndex;
145     Skip = Arithmetic.Decrement(ref currentInternalReferenceIndex);
146     Any = Arithmetic.Decrement(ref currentInternalReferenceIndex);
147     Itself = Arithmetic.Decrement(ref currentInternalReferenceIndex);
148     Arithmetic.Decrement(ref currentInternalReferenceIndex);
149     InternalReferencesRange = (possibleInternalReferencesRange.Minimum,
150     ↪ currentInternalReferenceIndex);
151     ExternalReferencesRange = possibleExternalReferencesRange;
152 }
153
154
155 /// <summary>
156 /// <para>
157 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
158 /// </para>
159 /// </summary>
160 /// <param name="targetPart">
161 /// <para>A target part.</para>
162 /// </param>
163 /// <param name="enableExternalReferencesSupport">
164 /// <para>A enable external references support.</para>
165 /// </param>
166 [MethodImpl(MethodImplOptions.AggressiveInlining)]
167 public LinksConstants(int targetPart, bool enableExternalReferencesSupport) :
168     ↪ this(targetPart, GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
169     ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
170
171
172 /// <summary>
173 /// <para>
174 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
175 /// </para>
176 /// </summary>
177 /// <param name="possibleInternalReferencesRange">
178 /// <para>A possible internal references range.</para>
179 /// </param>
180 /// <param name="possibleExternalReferencesRange">
181 /// <para>A possible external references range.</para>
182

```

```

184 /// <para></para>
185 /// </param>
186 [MethodImpl(MethodImplOptions.AggressiveInlining)]
187 public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange,
    ↳ Range<TLinkAddress>? possibleExternalReferencesRange) : this(DefaultTargetPart,
    ↳ possibleInternalReferencesRange, possibleExternalReferencesRange) { }
188
189 /// <summary>
190 /// <para>
191 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
192 /// </para>
193 /// <para></para>
194 /// </summary>
195 /// <param name="enableExternalReferencesSupport">
196 /// <para>A enable external references support.</para>
197 /// <para></para>
198 /// </param>
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 public LinksConstants(bool enableExternalReferencesSupport) :
    ↳ this(GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
    ↳ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
201
202 /// <summary>
203 /// <para>
204 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <param name="targetPart">
209 /// <para>A target part.</para>
210 /// <para></para>
211 /// </param>
212 /// <param name="possibleInternalReferencesRange">
213 /// <para>A possible internal references range.</para>
214 /// <para></para>
215 /// </param>
216 [MethodImpl(MethodImplOptions.AggressiveInlining)]
217 public LinksConstants(int targetPart, Range<TLinkAddress>
    ↳ possibleInternalReferencesRange) : this(targetPart, possibleInternalReferencesRange,
    ↳ null) { }
218
219 /// <summary>
220 /// <para>
221 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
222 /// </para>
223 /// <para></para>
224 /// </summary>
225 /// <param name="possibleInternalReferencesRange">
226 /// <para>A possible internal references range.</para>
227 /// <para></para>
228 /// </param>
229 [MethodImpl(MethodImplOptions.AggressiveInlining)]
230 public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange) :
    ↳ this(DefaultTargetPart, possibleInternalReferencesRange, null) { }
231
232 /// <summary>
233 /// <para>
234 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
235 /// </para>
236 /// <para></para>
237 /// </summary>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 public LinksConstants() : this(DefaultTargetPart, enableExternalReferencesSupport:
    ↳ false) { }
240
241 /// <summary>
242 /// <para>
243 /// Gets the default internal references range using the specified enable external
    ↳ references support.
244 /// </para>
245 /// <para></para>
246 /// </summary>
247 /// <param name="enableExternalReferencesSupport">
248 /// <para>The enable external references support.</para>
249 /// <para></para>
250 /// </param>
251 /// </returns>

```



```

252     /// <para>A range of t link address</para>
253     /// <para></para>
254     /// </returns>
255     [MethodImpl(MethodImplOptions.AggressiveInlining)]
256     public static Range<TLinkAddress> GetDefaultInternalReferencesRange(bool
    ↪ enableExternalReferencesSupport)
257     {
258         if (enableExternalReferencesSupport)
259         {
260             return (_one, _uInt64ToAddressConverter.Convert(Hybrid<TLinkAddress>.HalfOfNumbe
    ↪ rValuesRange));
261         }
262         else
263         {
264             return (_one, NumericType<TLinkAddress>.MaxValue);
265         }
266     }
267
268     /// <summary>
269     /// <para>
270     /// Gets the default external references range using the specified enable external
    ↪ references support.
271     /// </para>
272     /// <para></para>
273     /// </summary>
274     /// <param name="enableExternalReferencesSupport">
275     /// <para>The enable external references support.</para>
276     /// <para></para>
277     /// </param>
278     /// <returns>
279     /// <para>A range of t link address</para>
280     /// <para></para>
281     /// </returns>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     public static Range<TLinkAddress>? GetDefaultExternalReferencesRange(bool
    ↪ enableExternalReferencesSupport)
284     {
285         if (enableExternalReferencesSupport)
286         {
287             return (Hybrid<TLinkAddress>.ExternalZero, NumericType<TLinkAddress>.MaxValue);
288         }
289         else
290         {
291             return null;
292         }
293     }
294 }
295 }

```

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

```

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

```

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

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  using System.Runtime.CompilerServices;
4
5  namespace Platform.Data

```

```

6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links constants extensions.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public static class LinksConstantsExtensions
14    {
15        /// <summary>
16        /// <para>
17        /// Determines whether is reference.
18        /// </para>
19        /// <para></para>
20        /// </summary>
21        /// <typeparam name="TLinkAddress">
22        /// <para>The link address.</para>
23        /// <para></para>
24        /// </typeparam>
25        /// <param name="linksConstants">
26        /// <para>The links constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="address">
30        /// <para>The address.</para>
31        /// <para></para>
32        /// </param>
33        /// <returns>
34        /// <para>The bool</para>
35        /// <para></para>
36        /// </returns>
37        [MethodImpl(MethodImplOptions.AggressiveInlining)]
38        public static bool IsReference<TLinkAddress>(this LinksConstants<TLinkAddress>
39            ↪ linksConstants, TLinkAddress address) => linksConstants.IsInternalReference(address)
40            ↪ || linksConstants.IsExternalReference(address);
41
42        /// <summary>
43        /// <para>
44        /// Determines whether is internal reference.
45        /// </para>
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        [MethodImpl(MethodImplOptions.AggressiveInlining)]
65        public static bool IsInternalReference<TLinkAddress>(this LinksConstants<TLinkAddress>
66            ↪ linksConstants, TLinkAddress address) =>
67            ↪ linksConstants.InternalReferencesRange.Contains(address);
68
69        /// <summary>
70        /// <para>
71        /// Determines whether is external reference.
72        /// </para>
73        /// <para></para>
74        /// </summary>
75        /// <typeparam name="TLinkAddress">
76        /// <para>The link address.</para>
77        /// <para></para>
78        /// </typeparam>
79        /// <param name="linksConstants">
80        /// <para>The links constants.</para>
81        /// <para></para>
82        /// </param>
83        /// <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 =
    ↪ UncheckedConverter<long, TLink>.Default;
24
25         /// <summary>
26         /// <para>

```

```

27     /// Converts the source.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     /// <param name="source">
32     /// <para>The source.</para>
33     /// <para></para>
34     /// </param>
35     /// <returns>
36     /// <para>The link</para>
37     /// <para></para>
38     /// </returns>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public TLink Convert(TLink source) => _converter.Convert(new
        ↳ Hybrid<TLink>(source).AbsoluteValue);
41 }
42 }

```

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

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Exceptions;
6  using Platform.Ranges;
7  using Platform.Collections;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the point.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="IEquatable{LinkAddress{TLinkAddress}}"/>
20     /// <seealso cref="IList{TLinkAddress}"/>
21     public class Point<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>, IList<TLinkAddress>
22     {
23         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
24             ↳ EqualityComparer<TLinkAddress>.Default;
25
26         /// <summary>
27         /// <para>
28         /// Gets the index value.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         public TLinkAddress Index
33         {
34             [MethodImpl(MethodImplOptions.AggressiveInlining)]
35             get;
36         }
37
38         /// <summary>
39         /// <para>
40         /// Gets the size value.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         public int Size
45         {
46             [MethodImpl(MethodImplOptions.AggressiveInlining)]
47             get;
48         }
49
50         /// <summary>
51         /// <para>
52         /// The not supported exception.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         public TLinkAddress this[int index]
57         {
58             [MethodImpl(MethodImplOptions.AggressiveInlining)]
59             get
60             {

```

```

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

```

```

139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance contains.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="item">
147     /// <para>The item.</para>
148     /// <para></para>
149     /// </param>
150     /// <returns>
151     /// <para>The bool</para>
152     /// <para></para>
153     /// </returns>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     public virtual bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index);
156
157     /// <summary>
158     /// <para>
159     /// Copies the to using the specified array.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     /// <param name="array">
164     /// <para>The array.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="arrayIndex">
168     /// <para>The array index.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
173
174     /// <summary>
175     /// <para>
176     /// Gets the enumerator.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <returns>
181     /// <para>An enumerator of t link address</para>
182     /// <para></para>
183     /// </returns>
184     [MethodImpl(MethodImplOptions.AggressiveInlining)]
185     public IEnumerator<TLinkAddress> GetEnumerator()
186     {
187         for (int i = 0; i < Size; i++)
188         {
189             yield return Index;
190         }
191     }
192
193     /// <summary>
194     /// <para>
195     /// Indexes the of using the specified item.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="item">
200     /// <para>The item.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The int</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
209         ↪ 0 : -1;
210
211     /// <summary>
212     /// <para>
213     /// Inserts the index.
214     /// </para>
215     /// <para></para>
216     /// </summary>

```

```

216     /// <param name="index">
217     /// <para>The index.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="item">
221     /// <para>The item.</para>
222     /// <para></para>
223     /// </param>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
226
227     /// <summary>
228     /// <para>
229     /// Determines whether this instance remove.
230     /// </para>
231     /// <para></para>
232     /// </summary>
233     /// <param name="item">
234     /// <para>The item.</para>
235     /// <para></para>
236     /// </param>
237     /// <returns>
238     /// <para>The bool</para>
239     /// <para></para>
240     /// </returns>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     public bool Remove(TLinkAddress item) => throw new NotSupportedException();
243
244     /// <summary>
245     /// <para>
246     /// Removes the at using the specified index.
247     /// </para>
248     /// <para></para>
249     /// </summary>
250     /// <param name="index">
251     /// <para>The index.</para>
252     /// <para></para>
253     /// </param>
254     [MethodImpl(MethodImplOptions.AggressiveInlining)]
255     public void RemoveAt(int index) => throw new NotSupportedException();
256
257     /// <summary>
258     /// <para>
259     /// Gets the enumerator.
260     /// </para>
261     /// <para></para>
262     /// </summary>
263     /// <returns>
264     /// <para>The enumerator</para>
265     /// <para></para>
266     /// </returns>
267     [MethodImpl(MethodImplOptions.AggressiveInlining)]
268     IEnumerator IEnumerable.GetEnumerator()
269     {
270         for (int i = 0; i < Size; i++)
271         {
272             yield return Index;
273         }
274     }
275
276     /// <summary>
277     /// <para>
278     /// Determines whether this instance equals.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <param name="other">
283     /// <para>The other.</para>
284     /// <para></para>
285     /// </param>
286     /// <returns>
287     /// <para>The bool</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
        ⇨ _equalityComparer.Equals(Index, other.Index);

```

```

293 [MethodImpl(MethodImplOptions.AggressiveInlining)]
294 public static implicit operator TLinkAddress(Point<TLinkAddress> linkAddress) =>
    ↳ linkAddress.Index;

295
296 /// <summary>
297 /// <para>
298 /// Determines whether this instance equals.
299 /// </para>
300 /// <para></para>
301 /// </summary>
302 /// <param name="obj">
303 /// <para>The obj.</para>
304 /// <para></para>
305 /// </param>
306 /// <returns>
307 /// <para>The bool</para>
308 /// <para></para>
309 /// </returns>
310 [MethodImpl(MethodImplOptions.AggressiveInlining)]
311 public override bool Equals(object obj) => obj is Point<TLinkAddress> linkAddress ?
    ↳ Equals(linkAddress) : false;

312
313 /// <summary>
314 /// <para>
315 /// Gets the hash code.
316 /// </para>
317 /// <para></para>
318 /// </summary>
319 /// <returns>
320 /// <para>The int</para>
321 /// <para></para>
322 /// </returns>
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 public override int GetHashCode() => Index.GetHashCode();

325
326 /// <summary>
327 /// <para>
328 /// Returns the string.
329 /// </para>
330 /// <para></para>
331 /// </summary>
332 /// <returns>
333 /// <para>The string</para>
334 /// <para></para>
335 /// </returns>
336 [MethodImpl(MethodImplOptions.AggressiveInlining)]
337 public override string ToString() => Index.ToString();

338
339 [MethodImpl(MethodImplOptions.AggressiveInlining)]
340 public static bool operator ==(Point<TLinkAddress> left, Point<TLinkAddress> right)
341 {
342     if (left == null && right == null)
343     {
344         return true;
345     }
346     if (left == null)
347     {
348         return false;
349     }
350     return left.Equals(right);
351 }

352
353 [MethodImpl(MethodImplOptions.AggressiveInlining)]
354 public static bool operator !=(Point<TLinkAddress> left, Point<TLinkAddress> right) =>
    ↳ !(left == right);

355
356 /// <summary>
357 /// <para>
358 /// Determines whether is full point.
359 /// </para>
360 /// <para></para>
361 /// </summary>
362 /// <param name="link">
363 /// <para>The link.</para>
364 /// <para></para>
365 /// </param>
366 /// <returns>
367 /// <para>The bool</para>

```



```

368     /// <para></para>
369     /// </returns>
370     [MethodImpl(MethodImplOptions.AggressiveInlining)]
371     public static bool IsFullPoint(params TLinkAddress[] link) =>
372         ↪ IsFullPoint((IList<TLinkAddress>)link);
373
374     /// <summary>
375     /// <para>
376     /// Determines whether is full point.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <param name="link">
381     /// <para>The link.</para>
382     /// </param>
383     /// <returns>
384     /// <para>The bool</para>
385     /// <para></para>
386     /// </returns>
387     [MethodImpl(MethodImplOptions.AggressiveInlining)]
388     public static bool IsFullPoint(IList<TLinkAddress> link)
389     {
390         Ensure.Always.ArgumentNotEmpty(link, nameof(link));
391         Ensure.Always.ArgumentInRange(link.Count, (2, int.MaxValue), nameof(link), "Cannot
392         ↪ determine link's pointness using only its identifier.");
393         return IsFullPointUnchecked(link);
394     }
395
396     /// <summary>
397     /// <para>
398     /// Determines whether is full point unchecked.
399     /// </para>
400     /// <para></para>
401     /// </summary>
402     /// <param name="link">
403     /// <para>The link.</para>
404     /// </param>
405     /// <returns>
406     /// <para>The result.</para>
407     /// <para></para>
408     /// </returns>
409     [MethodImpl(MethodImplOptions.AggressiveInlining)]
410     public static bool IsFullPointUnchecked(IList<TLinkAddress> link)
411     {
412         var result = true;
413         for (var i = 1; result && i < link.Count; i++)
414         {
415             result = _equalityComparer.Equals(link[0], link[i]);
416         }
417         return result;
418     }
419
420     /// <summary>
421     /// <para>
422     /// Determines whether is partial point.
423     /// </para>
424     /// <para></para>
425     /// </summary>
426     /// <param name="link">
427     /// <para>The link.</para>
428     /// </param>
429     /// <returns>
430     /// <para>The bool</para>
431     /// <para></para>
432     /// </returns>
433     [MethodImpl(MethodImplOptions.AggressiveInlining)]
434     public static bool IsPartialPoint(params TLinkAddress[] link) =>
435         ↪ IsPartialPoint((IList<TLinkAddress>)link);
436
437     /// <summary>
438     /// <para>
439     /// Determines whether is partial point.
440     /// </para>
441     /// <para></para>
442     /// </summary>

```

```

443     /// <param name="link">
444     /// <para>The link.</para>
445     /// <para></para>
446     /// </param>
447     /// <returns>
448     /// <para>The bool</para>
449     /// <para></para>
450     /// </returns>
451     [MethodImpl(MethodImplOptions.AggressiveInlining)]
452     public static bool IsPartialPoint(IList<TLinkAddress> link)
453     {
454         Ensure.Always.ArgumentNotEmpty(link, nameof(link));
455         Ensure.Always.ArgumentInRange(link.Count, (2, int.MaxValue), nameof(link), "Cannot
456         ↪ determine link's pointness using only its identifier.");
457         return IsPartialPointUnchecked(link);
458     }
459     /// <summary>
460     /// <para>
461     /// Determines whether is partial point unchecked.
462     /// </para>
463     /// <para></para>
464     /// </summary>
465     /// <param name="link">
466     /// <para>The link.</para>
467     /// <para></para>
468     /// </param>
469     /// <returns>
470     /// <para>The result.</para>
471     /// <para></para>
472     /// </returns>
473     [MethodImpl(MethodImplOptions.AggressiveInlining)]
474     public static bool IsPartialPointUnchecked(IList<TLinkAddress> link)
475     {
476         var result = false;
477         for (var i = 1; !result && i < link.Count; i++)
478         {
479             result = _equalityComparer.Equals(link[0], link[i]);
480         }
481         return result;
482     }
483 }
484 }

```

### 1.17 ./csharp/Platform.Data/ReadHandlerState.cs

```

1  using System.Collections.Generic;
2  using Platform.Delegates;
3
4  namespace Platform.Data
5  {
6      public struct ReadHandlerState<TLink>
7      {
8          private readonly EqualityComparer<TLink> _equalityComparer;
9          public TLink Result;
10         public ReadHandler<TLink> Handler;
11         public TLink Continue;
12         public TLink Break;
13
14         public ReadHandlerState(TLink @continue, TLink @break, ReadHandler<TLink> handler)
15         {
16             _equalityComparer = EqualityComparer<TLink>.Default;
17             Continue = @continue;
18             Break = @break;
19             Result = @continue;
20             Handler = handler;
21         }
22
23         public void Apply(TLink result)
24         {
25             if (_equalityComparer.Equals(Break, Result))
26             {
27                 return;
28             }
29             if (!_equalityComparer.Equals(Break, result))
30             {
31                 return;
32             }
33             Handler = null;
34             Result = Break;
35         }
36     }
37 }

```

```

36     }
37 }

```

## 1.18 ./csharp/Platform.Data/Universal/IUniLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Delegates;
4
5  // ReSharper disable TypeParameterCanBeVariant
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Universal
9  {
10     /// <remarks>Minimal sufficient universal Links API (for bulk operations).</remarks>
11     public partial interface IUniLinks<TLinkAddress>
12     {
13         /// <summary>
14         /// <para>
15         /// Triggers the condition.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         /// <param name="condition">
20         /// <para>The condition.</para>
21         /// <para></para>
22         /// </param>
23         /// <param name="substitution">
24         /// <para>The substitution.</para>
25         /// <para></para>
26         /// </param>
27         /// <returns>
28         /// <para>A list of i list i list t link address</para>
29         /// <para></para>
30         /// </returns>
31         IList<IList<IList<TLinkAddress>>> Trigger(IList<TLinkAddress> condition,
32             ↳ IList<TLinkAddress> substitution);
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, ReadHandler<TLinkAddress>
48             ↳ matchHandler,
49             ↳ IList<TLinkAddress> substitution, WriteHandler<TLinkAddress>
50             ↳ 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>

```

```

71     /// <returns>
72     /// <para>The link address</para>
73     /// <para></para>
74     /// </returns>
75     TLinkAddress Trigger(IList<TLinkAddress> restriction, WriteHandler<TLinkAddress>
    ↪     matchedHandler,
76         IList<TLinkAddress> substitution, WriteHandler<TLinkAddress> substitutedHandler);
77 }
78
79 /// <remarks>Extended with small optimization.</remarks>
80 public partial interface IUniLinks<TLinkAddress>
81 {
82     /// <remarks>
83     /// Something simple should be simple and optimized.
84     /// </remarks>
85     TLinkAddress Count(IList<TLinkAddress> restrictions);
86 }
87 }

```

### 1.19 ./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     TLinkAddress Delete(IList<TLinkAddress> parts);
97 }
98 }

```

## 1.20 ./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 }

```

## 1.21 ./csharp/Platform.Data/Universal/IUniLinksIO.cs

```

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

```

```

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

```

## 1.22 ./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
    ↪ default(TLinkAddress).
20         ///
21         /// Outs(returns) inner contents of link, its part/parent/element/value.
22         /// </remarks>
23         TLinkAddress OutOne(int partType, IList<TLinkAddress> pattern);
24
25         /// <remarks>OutCount() returns total links in store as array.</remarks>
26         IList<IList<TLinkAddress>> OutAll(IList<TLinkAddress> pattern);
27
28         /// <remarks>OutCount() returns total amount of links in store.</remarks>
29         ulong OutCount(IList<TLinkAddress> pattern);
30     }
31 }

```

## 1.23 ./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 }

```

#### 1.24 ./csharp/Platform.Data/WriteHandlerState.cs

```

1  using System.Collections.Generic;
2  using Platform.Delegates;
3
4  namespace Platform.Data
5  {
6      public struct WriteHandlerState<TLink>
7      {
8          private readonly EqualityComparer<TLink> _equalityComparer;
9          public TLink Result;
10         public WriteHandler<TLink> Handler;
11         public TLink Continue;
12         public TLink Break;
13
14         public WriteHandlerState(TLink @continue, TLink @break, WriteHandler<TLink> handler)
15         {
16             _equalityComparer = EqualityComparer<TLink>.Default;
17             Continue = @continue;
18             Break = @break;
19             Result = @continue;
20             Handler = handler;
21         }
22
23         public void Apply(TLink result)
24         {
25             if (_equalityComparer.Equals(Break, Result))
26             {
27                 return;
28             }
29             if (!_equalityComparer.Equals(Break, result))
30             {
31                 return;
32             }
33             Handler = null;

```



```

34         Result = Break;
35     }
36 }
37 }

```

## 1.25 ./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 }

```

## 1.26 ./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
31         /// <summary>
32         /// <para>
33         /// Tests that external references test.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         [Fact]
38         public static void ExternalReferencesTest()

```

```

38     {
39         TestExternalReferences<ulong, long>();
40         TestExternalReferences<uint, int>();
41         TestExternalReferences<ushort, short>();
42         TestExternalReferences<byte, sbyte>();
43     }
44     private static void TestExternalReferences<TUnsigned, TSigned>()
45     {
46         var unsingedOne = Arithmetic.Increment(default(TUnsigned));
47         var converter = UncheckedConverter<TSigned, TUnsigned>.Default;
48         var half = converter.Convert(NumericType<TSigned>.MaxValue);
49         LinksConstants<TUnsigned> constants = new LinksConstants<TUnsigned>((unsingedOne,
50             ↪ half), (Arithmetic.Add(half, unsingedOne), NumericType<TUnsigned>.MaxValue));
51
52         var minimum = new Hybrid<TUnsigned>(default, isExternal: true);
53         var maximum = new Hybrid<TUnsigned>(half, isExternal: true);
54
55         Assert.True(constants.IsExternalReference(minimum));
56         Assert.True(minimum.IsExternal);
57         Assert.False(minimum.IsInternal);
58         Assert.True(constants.IsExternalReference(maximum));
59         Assert.True(maximum.IsExternal);
60         Assert.False(maximum.IsInternal);
61     }
62 }

```

## Index

- ./csharp/Platform.Data.Tests/HybridTests.cs, 41
- ./csharp/Platform.Data.Tests/LinksConstantsTests.cs, 41
- ./csharp/Platform.Data/Exceptions/ArgumentLinkDoesNotExistException.cs, 1
- ./csharp/Platform.Data/Exceptions/ArgumentLinkHasDependenciesException.cs, 2
- ./csharp/Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsException.cs, 3
- ./csharp/Platform.Data/Exceptions/LinksLimitReachedException.cs, 4
- ./csharp/Platform.Data/Exceptions/LinksLimitReachedExceptionBase.cs, 5
- ./csharp/Platform.Data/Hybrid.cs, 6
- ./csharp/Platform.Data/ILinks.cs, 11
- ./csharp/Platform.Data/ILinksExtensions.cs, 13
- ./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, 25
- ./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/ReadHandlerState.cs, 34
- ./csharp/Platform.Data/Universal/IUniLinks.cs, 35
- ./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
- ./csharp/Platform.Data/WriteHandlerState.cs, 40