

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) =>
    ↪ (short)hybrid.AbsoluteValue;

251
252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
253 public static explicit operator byte(Hybrid<TLinkAddress> hybrid) =>
    ↪ CheckedConverter<TLinkAddress, byte>.Default.Convert(hybrid.Value);

254
255 [MethodImpl(MethodImplOptions.AggressiveInlining)]
256 public static explicit operator sbyte(Hybrid<TLinkAddress> hybrid) =>
    ↪ (sbyte)hybrid.AbsoluteValue;

257
258 /// <summary>
259 /// <para>
260 /// Returns the string.
261 /// </para>
262 /// <para></para>
263 /// </summary>
264 /// <returns>
265 /// <para>The string</para>
266 /// <para></para>
267 /// </returns>
268 [MethodImpl(MethodImplOptions.AggressiveInlining)]
269 public override string ToString() => IsExternal ? $"{<AbsoluteValue>}" :
    ↪ Value.ToString();

270
271 /// <summary>
272 /// <para>
273 /// Determines whether this instance equals.
274 /// </para>
275 /// <para></para>
276 /// </summary>
277 /// <param name="other">
278 /// <para>The other.</para>
279 /// <para></para>
280 /// </param>
281 /// <returns>
282 /// <para>The bool</para>
283 /// <para></para>
284 /// </returns>
285 [MethodImpl(MethodImplOptions.AggressiveInlining)]
286 public bool Equals(Hybrid<TLinkAddress> other) => _equalityComparer.Equals(Value,
    ↪ other.Value);

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

304
305 /// <summary>
306 /// <para>
307 /// Gets the hash code.
308 /// </para>
309 /// <para></para>
310 /// </summary>
311 /// <returns>
312 /// <para>The int</para>
313 /// <para></para>
314 /// </returns>
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 public override int GetHashCode() => Value.GetHashCode();

317
318 [MethodImpl(MethodImplOptions.AggressiveInlining)]
319 public static bool operator ==(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
    ↪ left.Equals(right);

320

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data
8  {
9      /// <summary>
10     /// <para>Represents an interface for manipulating data in the Links (links storage)
11     ↪     format.</para>
12     /// <para>Представляет интерфейс для манипуляции с данными в формате Links (хранилища
13     ↪     связей).</para>
14     /// </summary>
15     /// <remarks>
16     /// <para>This interface is independent of the size of the content of the link, meaning it
17     ↪     is suitable for both doublets, triplets, and link sequences of any size.</para>
18     /// <para>Этот интерфейс не зависит от размера содержимого связи, а значит подходит как для
19     ↪     дуплетов, триплетов и последовательностей связей любого размера.</para>
20     /// </remarks>
21     public interface ILinks<TLinkAddress, TConstants>
22     where TConstants : LinksConstants<TLinkAddress>
23     {
24         #region Constants
25
26         /// <summary>
27         /// <para>Returns the set of constants that is necessary for effective communication
28         ↪     with the methods of this interface.</para>
29         /// <para>Возвращает набор констант, который необходим для эффективной коммуникации с
30         ↪     методами этого интерфейса.</para>
31         /// </summary>
32         /// <remarks>
33         /// <para>These constants are not changed since the creation of the links storage access
34         ↪     point.</para>
35         /// <para>Эти константы не меняются с момента создания точки доступа к хранилищу
36         ↪     связей.</para>
37         /// </remarks>
38         TConstants Constants
39         {
40             [MethodImpl(MethodImplOptions.AggressiveInlining)]
41             get;
42         }
43
44         #endregion
45
46         #region Read
47
48         /// <summary>
49         /// <para>Counts and returns the total number of links in the storage that meet the
50         ↪     specified restriction.</para>
51         /// <para>Подсчитывает и возвращает общее число связей находящихся в хранилище,
52         ↪     соответствующих указанному ограничению.</para>
53         /// </summary>
54         /// <param name="restriction"><para>Restriction on the contents of
55         ↪     links.</para><para>Ограничение на содержимое связей.</para></param>
56         /// <returns><para>The total number of links in the storage that meet the specified
57         ↪     restriction.</para><para>Общее число связей находящихся в хранилище, соответствующих
58         ↪     указанному ограничению.</para></returns>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         TLinkAddress Count(IList<TLinkAddress> restriction);
61
62         /// <summary>
63         /// <para>Passes through all the links matching the pattern, invoking a handler for each
64         ↪     matching link.</para>
65         /// <para>Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
66         ↪     (handler) для каждой подходящей связи.</para>
67         /// </summary>
68         /// <param name="restriction">
69         /// <para>Restriction on the contents of links. Each constraint can have values:
70         ↪     Constants.Null - the 0th link denoting a reference to the void, Any - the absence of
71         ↪     a constraint, 1..∞ a specific link index.</para>

```

```

55  /// <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>
56  /// </param>
57  /// <param name="handler"><para>A handler for each matching link.</para><para>Обработчик
    → для каждой подходящей связи.</para></param>
58  /// <returns><para>Constants.Continue, if the pass through the links was not
    → interrupted, and Constants.Break otherwise.</para><para>Constants.Continue, в случае
    → если проход по связям не был прерван и Constants.Break в обратном
    → случае.</para></returns>
59  [MethodImpl(MethodImplOptions.AggressiveInlining)]
60  TLinkAddress Each(IList<TLinkAddress> restriction, Func<IList<TLinkAddress>,
    → TLinkAddress> handler);
61
62  #endregion
63
64  #region Write
65
66  /// <summary>
67  /// <para>Creates a link.</para>
68  /// <para>Создаёт связь.</para>
69  /// <param name="substitution">
70  /// <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>
71  /// <para>Содержимое новой связи. Этот аргумент опционален, если null передан в качестве
    → значения это означает, что никакого содержимого для связи не установлено.</para>
72  /// </param>
73  /// <param name="handler">
74  /// <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>
75  /// <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>
76  /// </param>
77  /// </summary>
78  /// <returns>
79  /// <para>
80  /// Constants.Continue if all executed changes are handled.
81  /// Constants.Break if proccessing of handled changes is stoped.
82  /// </para>
83  /// <para>
84  /// Constants.Continue если все выполненные изменения обработаны.
85  /// Constants.Break если обработка выполненных изменений остановлена.
86  /// </para>
87  /// </returns>
88  [MethodImpl(MethodImplOptions.AggressiveInlining)]
89  TLinkAddress Create(IList<TLinkAddress> substitution, Func<IList<TLinkAddress>,
    → IList<TLinkAddress>, TLinkAddress> handler);
90
91  /// <summary>
92  /// Обновляет связь с указанными restriction[Constants.IndexPart] в адресом связи
93  /// на связь с указанным новым содержимым.
94  /// </summary>
95  /// <param name="restriction">
96  /// Ограничение на содержимое связей.
97  /// Предполагается, что будет указан индекс связи (в restriction[Constants.IndexPart]) и
    → далее за ним будет следовать содержимое связи.
98  /// Каждое ограничение может иметь значения: Constants.Null - 0-я связь, обозначающая
    → ссылку на пустоту,
99  /// Constants.Itself - требование установить ссылку на себя, 1.. $\infty$  конкретный индекс
    → другой связи.
100  /// </param>
101  /// <param name="substitution"></param>
102  /// <param name="handler">
103  /// <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>
104  /// <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>
105  /// </param>
106  /// <returns>
107  /// <para>
108  /// Constants.Continue if all executed changes are handled.

```

```

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

```

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

```

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

```

```

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

```

```

97  /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
    ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
    ↳ Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
98  /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
99  [MethodImpl(MethodImplOptions.AggressiveInlining)]
100 public static TLinkAddress Each<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, Func<IList<TLinkAddress>, TLinkAddress> handler, params
    ↳ TLinkAddress[] restrictions)
101     where TConstants : LinksConstants<TLinkAddress>
102     => links.Each(handler, restrictions);
103
104  /// <summary>
105  /// Возвращает части-значения для связи с указанным индексом.
106  /// </summary>
107  /// <param name="links">Хранилище связей.</param>
108  /// <param name="link">Индекс связи.</param>
109  /// <returns>Уникальную связь.</returns>
110  [MethodImpl(MethodImplOptions.AggressiveInlining)]
111  public static IList<TLinkAddress> GetLink<TLinkAddress, TConstants>(this
    ↳ ILinks<TLinkAddress, TConstants> links, TLinkAddress link)
112     where TConstants : LinksConstants<TLinkAddress>
113  {
114     var constants = links.Constants;
115     if (constants.IsExternalReference(link))
116     {
117         return new Point<TLinkAddress>(link, constants.TargetPart + 1);
118     }
119     var linkPartsSetter = new Setter<IList<TLinkAddress>,
    ↳ TLinkAddress>(constants.Continue, constants.Break);
120     links.Each(linkPartsSetter.SetAndReturnTrue, link);
121     return linkPartsSetter.Result;
122 }
123
124 #region Points
125
126  /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
    ↳ точкой полностью (связью замкнутой на себе дважды).</summary>
127  /// <param name="links">Хранилище связей.</param>
128  /// <param name="link">Индекс проверяемой связи.</param>
129  /// <returns>Значение, определяющее является ли связь точкой полностью.</returns>
130  /// <remarks>
131  /// Связь точка - это связь, у которой начало (Source) и конец (Target) есть сама эта
    ↳ связь.
132  /// Но что, если точка уже есть, а нужно создать пару с таким же значением? Должны ли
    ↳ точка и пара существовать одновременно?
133  /// Или в качестве решения для точек нужно использовать 0 в качестве начала и конца, а
    ↳ сортировать по индексу в массиве связей?
134  /// Какое тогда будет значение Source и Target у точки? 0 или её индекс?
135  /// Или точка должна быть одновременно точкой и парой, а также последовательностями из
    ↳ самой себя любого размера?
136  /// Как только есть ссылка на себя, появляется этот парадокс, причём достаточно даже
    ↳ одной ссылки на себя (частичной точки).
137  /// А что если не выбирать что является точкой, пара нулей (цикл через пустоту) или
138  /// самостоятельный цикл через себя? Что если предоставить все варианты использования
    ↳ связей?
139  /// Что если разрешить и нули, а так же частичные варианты?
140  ///
141  /// Что если точка, это только в том случае когда link.Source == link &&
    ↳ link.Target == link , т.е. дважды ссылка на себя.
142  /// А пара это тогда, когда link.Source == link.Target && link.Source != link ,
    ↳ т.е. ссылка не на себя а во вне.
143  ///
144  /// Тогда если у нас уже создана пара, но нам нужна точка, мы можем используя
    ↳ промежуточную связь,
145  /// например "DoubletOf" обозначить что является точно парой, а что точно точкой.
146  /// И наоборот этот же метод поможет, если уже существует точка, но нам нужна пара.
147  /// </remarks>
148  [MethodImpl(MethodImplOptions.AggressiveInlining)]
149  public static bool IsFullPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, TLinkAddress link)
150     where TConstants : LinksConstants<TLinkAddress>
151  {
152     if (links.Constants.IsExternalReference(link))
153     {
154         return true;
155     }
156     links.EnsureLinkExists(link);

```

```

157         return Point<TLinkAddress>.IsFullPoint(links.GetLink(link));
158     }
159
160     /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
161     ↪ точкой частично (связью замкнутой на себе как минимум один раз).</summary>
162     /// <param name="links">Хранилище связей.</param>
163     /// <param name="link">Индекс проверяемой связи.</param>
164     /// <returns>Значение, определяющее является ли связь точкой частично.</returns>
165     /// <remarks>
166     /// Достаточно любой одной ссылки на себя.
167     /// Также в будущем можно будет проверять и всех родителей, чтобы проверить есть ли
168     ↪ ссылки на себя (на эту связь).
169     /// </remarks>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     public static bool IsPartialPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
172     ↪ TConstants> links, TLinkAddress link)
173     where TConstants : LinksConstants<TLinkAddress>
174     {
175         if (links.Constants.IsExternalReference(link))
176         {
177             return true;
178         }
179         links.EnsureLinkExists(link);
180         return Point<TLinkAddress>.IsPartialPoint(links.GetLink(link));
181     }
182 }
#endregion
}

```

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    }
21 }

```

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>

```



```

24     /// Gets the index value.
25     /// </para>
26     /// <para></para>
27     /// </summary>
28     public TLinkAddress Index
29     {
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         get;
32     }
33
34     /// <summary>
35     /// <para>
36     /// The not supported exception.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     public TLinkAddress this[int index]
41     {
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         get
44         {
45             if (index == 0)
46             {
47                 return Index;
48             }
49             else
50             {
51                 throw new IndexOutOfRangeException();
52             }
53         }
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         set => throw new NotSupportedException();
56     }
57
58     /// <summary>
59     /// <para>
60     /// Gets the count value.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     public int Count
65     {
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         get => 1;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Gets the is read only value.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public bool IsReadOnly
77     {
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         get => true;
80     }
81
82     /// <summary>
83     /// <para>
84     /// Initializes a new <see cref="LinkAddress{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 }

```

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

```

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

```

```

72 #endregion
73
74 #region Special symbols
75
76
77 /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
78 public TLinkAddress Null
79 {
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     get;
82 }
83
84 /// <summary>Возвращает значение, обозначающее любую связь.</summary>
85 /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
86     ↳ создавать все варианты последовательностей в функции Create.</remarks>
87 public TLinkAddress Any
88 {
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     get;
91 }
92
93 /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
94 public TLinkAddress Itself
95 {
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     get;
98 }
99 #endregion
100
101 #region References
102
103 /// <summary>Возвращает диапазон возможных индексов для внутренних связей (внутренних
104     ↳ ссылок).</summary>
105 public Range<TLinkAddress> InternalReferencesRange
106 {
107     [MethodImpl(MethodImplOptions.AggressiveInlining)]
108     get;
109 }
110
111 /// <summary>Возвращает диапазон возможных индексов для внешних связей (внешних
112     ↳ ссылок).</summary>
113 public Range<TLinkAddress>? ExternalReferencesRange
114 {
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     get;
117 }
118 #endregion
119
120 /// <summary>
121 /// <para>
122 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
123 /// </para>
124 /// </summary>
125 /// <param name="targetPart">
126 /// <para>A target part.</para>
127 /// </param>
128 /// <param name="possibleInternalReferencesRange">
129 /// <para>A possible internal references range.</para>
130 /// </param>
131 /// <param name="possibleExternalReferencesRange">
132 /// <para>A possible external references range.</para>
133 /// </param>
134 [MethodImpl(MethodImplOptions.AggressiveInlining)]
135 public LinksConstants(int targetPart, Range<TLinkAddress>
136     ↳ possibleInternalReferencesRange, Range<TLinkAddress>?
137     ↳ possibleExternalReferencesRange)
138 {
139     IndexPart = 0;
140     SourcePart = 1;
141     TargetPart = targetPart;
142     Null = default;
143     Break = default;
144     var currentInternalReferenceIndex = possibleInternalReferencesRange.Maximum;
145     Continue = currentInternalReferenceIndex;
146

```

```

147     Skip = Arithmetic.Decrement(ref currentInternalReferenceIndex);
148     Any = Arithmetic.Decrement(ref currentInternalReferenceIndex);
149     Itself = Arithmetic.Decrement(ref currentInternalReferenceIndex);
150     Arithmetic.Decrement(ref currentInternalReferenceIndex);
151     InternalReferencesRange = (possibleInternalReferencesRange.Minimum,
        ↳ currentInternalReferenceIndex);
152     ExternalReferencesRange = possibleExternalReferencesRange;
153 }
154
155 /// <summary>
156 /// <para>
157 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
158 /// </para>
159 /// <para></para>
160 /// </summary>
161 /// <param name="targetPart">
162 /// <para>A target part.</para>
163 /// <para></para>
164 /// </param>
165 /// <param name="enableExternalReferencesSupport">
166 /// <para>A enable external references support.</para>
167 /// <para></para>
168 /// </param>
169 [MethodImpl(MethodImplOptions.AggressiveInlining)]
170 public LinksConstants(int targetPart, bool enableExternalReferencesSupport) :
    ↳ this(targetPart, GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
    ↳ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
171
172 /// <summary>
173 /// <para>
174 /// Initializes a new <see cref="LinksConstants{TLinkAddress}"/> instance.
175 /// </para>
176 /// <para></para>
177 /// </summary>
178 /// <param name="possibleInternalReferencesRange">
179 /// <para>A possible internal references range.</para>
180 /// <para></para>
181 /// </param>
182 /// <param name="possibleExternalReferencesRange">
183 /// <para>A possible external references range.</para>
184 /// <para></para>
185 /// </param>
186 [MethodImpl(MethodImplOptions.AggressiveInlining)]
187 public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange,
    ↳ Range<TLinkAddress>? possibleExternalReferencesRange) : this(DefaultTargetPart,
    ↳ possibleInternalReferencesRange, possibleExternalReferencesRange) { }
188
189 /// <summary>
190 /// <para>
191 /// Initializes a new <see cref="LinksConstants{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>

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

20  /// <seealso cref="IList{TLinkAddress}"/>
21  public class Point<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>, IList<TLinkAddress>
22  {
23      private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
24          ↳ EqualityComparer<TLinkAddress>.Default;
25
26      /// <summary>
27      /// <para>
28      /// Gets the index value.
29      /// </para>
30      /// <para></para>
31      /// </summary>
32      public TLinkAddress Index
33      {
34          [MethodImpl(MethodImplOptions.AggressiveInlining)]
35          get;
36      }
37
38      /// <summary>
39      /// <para>
40      /// Gets the size value.
41      /// </para>
42      /// <para></para>
43      /// </summary>
44      public int Size
45      {
46          [MethodImpl(MethodImplOptions.AggressiveInlining)]
47          get;
48      }
49
50      /// <summary>
51      /// <para>
52      /// The not supported exception.
53      /// </para>
54      /// <para></para>
55      /// </summary>
56      public TLinkAddress this[int index]
57      {
58          [MethodImpl(MethodImplOptions.AggressiveInlining)]
59          get
60          {
61              if (index < Size)
62              {
63                  return Index;
64              }
65              else
66              {
67                  throw new IndexOutOfRangeException();
68              }
69          }
70          [MethodImpl(MethodImplOptions.AggressiveInlining)]
71          set => throw new NotSupportedException();
72      }
73
74      /// <summary>
75      /// <para>
76      /// Gets the count value.
77      /// </para>
78      /// <para></para>
79      /// </summary>
80      public int Count
81      {
82          [MethodImpl(MethodImplOptions.AggressiveInlining)]
83          get => Size;
84      }
85
86      /// <summary>
87      /// <para>
88      /// Gets the is read only value.
89      /// </para>
90      /// <para></para>
91      /// </summary>
92      public bool IsReadOnly
93      {
94          [MethodImpl(MethodImplOptions.AggressiveInlining)]
95          get => true;
96      }
97
98      /// <summary>

```

```

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) ?
        0 : -1;
209
210     /// <summary>
211     /// <para>
212     /// Inserts the index.
213     /// </para>
214     /// <para></para>
215     /// </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);
292
293     [MethodImpl(MethodImplOptions.AggressiveInlining)]
294     public static implicit operator TLinkAddress(Point<TLinkAddress> linkAddress) =>
        ↪ linkAddress.Index;
295
296     /// <summary>
297     /// <para>
298     /// Determines whether this instance equals.
299     /// </para>
300     /// <para></para>
301     /// </summary>
302     /// <param name="obj">
303     /// <para>The obj.</para>
304     /// <para></para>
305     /// </param>
306     /// <returns>
307     /// <para>The bool</para>
308     /// <para></para>
309     /// </returns>
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     public override bool Equals(object obj) => obj is Point<TLinkAddress> linkAddress ?
        ↪ Equals(linkAddress) : false;
312
313     /// <summary>
314     /// <para>
315     /// Gets the hash code.
316     /// </para>
317     /// <para></para>
318     /// </summary>
319     /// <returns>
320     /// <para>The int</para>
321     /// <para></para>
322     /// </returns>
323     [MethodImpl(MethodImplOptions.AggressiveInlining)]
324     public override int GetHashCode() => Index.GetHashCode();
325
326     /// <summary>
327     /// <para>

```

```

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

```



```

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

```

```

479         result = _equalityComparer.Equals(link[0], link[i]);
480     }
481     return result;
482 }
483 }
484 }

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

29         Assert.Equal(1, new Hybrid<byte>((object)1).AbsoluteValue);
30     }
31 }
32 }

```

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

```

1  using Xunit;
2  using Platform.Reflection;
3  using Platform.Converters;
4  using Platform.Numbers;
5
6  namespace Platform.Data.Tests
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links constants tests.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     public static class LinksConstantsTests
15     {
16         /// <summary>
17         /// <para>
18         /// Tests that constructor test.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         [Fact]
23         public static void ConstructorTest()
24         {
25             var constants = new LinksConstants<ulong>(enableExternalReferencesSupport: true);
26             Assert.Equal(Hybrid<ulong>.ExternalZero,
27                 ↪ constants.ExternalReferencesRange.Value.Minimum);
28             Assert.Equal(ulong.MaxValue, constants.ExternalReferencesRange.Value.Maximum);
29         }
30
31         /// <summary>
32         /// <para>
33         /// Tests that external references test.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         [Fact]
38         public static void ExternalReferencesTest()
39         {
40             TestExternalReferences<ulong, long>();
41             TestExternalReferences<uint, int>();
42             TestExternalReferences<ushort, short>();
43             TestExternalReferences<byte, sbyte>();
44         }
45
46         private static void TestExternalReferences<TUnsigned, TSigned>()
47         {
48             var unsingedOne = Arithmetic.Increment(default(TUnsigned));
49             var converter = UncheckedConverter<TSigned, TUnsigned>.Default;
50             var half = converter.Convert(NumericType<TSigned>.MaxValue);
51             LinksConstants<TUnsigned> constants = new LinksConstants<TUnsigned>((unsingedOne,
52                 ↪ half), (Arithmetic.Add(half, unsingedOne), NumericType<TUnsigned>.MaxValue));
53
54             var minimum = new Hybrid<TUnsigned>(default, isExternal: true);
55             var maximum = new Hybrid<TUnsigned>(half, isExternal: true);
56
57             Assert.True(constants.IsExternalReference(minimum));
58             Assert.True(minimum.IsExternal);
59             Assert.False(minimum.IsInternal);
60             Assert.True(constants.IsExternalReference(maximum));
61             Assert.True(maximum.IsExternal);
62             Assert.False(maximum.IsInternal);
63         }
64     }
65 }
66 }

```


Index

- ./csharp/Platform.Data.Tests/HybridTests.cs, 39
- ./csharp/Platform.Data.Tests/LinksConstantsTests.cs, 40
- ./csharp/Platform.Data/Exceptions/ArgumentLinkDoesNotExistException.cs, 1
- ./csharp/Platform.Data/Exceptions/ArgumentLinkHasDependenciesException.cs, 2
- ./csharp/Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsException.cs, 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, 16
- ./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, 26
- ./csharp/Platform.Data/Numbers/Raw/RawNumberToAddressConverter.cs, 27
- ./csharp/Platform.Data/Point.cs, 27
- ./csharp/Platform.Data/Universal/IUniLinks.cs, 34
- ./csharp/Platform.Data/Universal/IUniLinksCRUD.cs, 35
- ./csharp/Platform.Data/Universal/IUniLinksGS.cs, 36
- ./csharp/Platform.Data/Universal/IUniLinksIO.cs, 37
- ./csharp/Platform.Data/Universal/IUniLinksIOWithExtensions.cs, 38
- ./csharp/Platform.Data/Universal/IUniLinksRW.cs, 38