

## LinksPlatform's Platform.Data Class Library

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

```
1 using System;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Exceptions
6 {
7     public class ArgumentLinkDoesNotExistsException<TLinkAddress> : ArgumentException
8     {
9         public ArgumentLinkDoesNotExistsException(TLinkAddress link, string argumentName) :
10             base(FormatMessage(link, argumentName), argumentName) { }
11
12         public ArgumentLinkDoesNotExistsException(TLinkAddress link) : base(FormatMessage(link))
13             { }
14
15         public ArgumentLinkDoesNotExistsException(string message, Exception innerException) :
16             base(message, innerException) { }
17
18         public ArgumentLinkDoesNotExistsException(string message) : base(message) { }
19
20         public ArgumentLinkDoesNotExistsException() { }
21
22         private static string FormatMessage(TLinkAddress link, string argumentName) => $"Связь
23             [{link}] переданная в аргумент [{argumentName}] не существует.";
24
25         private static string FormatMessage(TLinkAddress link) => $"Связь [{link}] переданная в
26             качестве аргумента не существует.";
27     }
28 }
```

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

```
1 using System;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Exceptions
6 {
7     public class ArgumentLinkHasDependenciesException<TLinkAddress> : ArgumentException
8     {
9         public ArgumentLinkHasDependenciesException(TLinkAddress link, string paramName) :
10             base(FormatMessage(link, paramName), paramName) { }
11
12         public ArgumentLinkHasDependenciesException(TLinkAddress link) :
13             base(FormatMessage(link)) { }
14
15         public ArgumentLinkHasDependenciesException(string message, Exception innerException) :
16             base(message, innerException) { }
17
18         public ArgumentLinkHasDependenciesException(string message) : base(message) { }
19
20         public ArgumentLinkHasDependenciesException() { }
21
22         private static string FormatMessage(TLinkAddress link, string paramName) => $"У связи
23             [{link}] переданной в аргумент [{paramName}] присутствуют зависимости, которые
24             препятствуют изменению её внутренней структуры.";
25
26         private static string FormatMessage(TLinkAddress link) => $"У связи [{link}] переданной
27             в качестве аргумента присутствуют зависимости, которые препятствуют изменению её
28             внутренней структуры.";
29     }
30 }
```

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

```
1 using System;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Exceptions
6 {
7     public class LinkWithSameValueAlreadyExistsException :
8         LinkWithSameValueAlreadyExistsExceptionBase
9     {
10         public LinkWithSameValueAlreadyExistsException(string message, Exception innerException)
11             : base(message, innerException) { }
12
13         public LinkWithSameValueAlreadyExistsException(string message) : base(message) { }
14
15         public LinkWithSameValueAlreadyExistsException() : base(DefaultMessage) { }
16     }
17 }
```

```
14     }
15 }
```

#### 1.4 ./Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsExceptionBase.cs

```
1 using System;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Exceptions
6 {
7     public abstract class LinkWithSameValueAlreadyExistsExceptionBase : Exception
8     {
9         public const string DefaultMessage = "Связь с таким же значением уже существует.";
10
11         protected LinkWithSameValueAlreadyExistsExceptionBase(string message, Exception
12             ↳ innerException) : base(message, innerException) { }
13
14         protected LinkWithSameValueAlreadyExistsExceptionBase(string message) : base(message) { }
15
16         protected LinkWithSameValueAlreadyExistsExceptionBase() { }
17     }
18 }
```

#### 1.5 ./Platform.Data/Exceptions/LinksLimitReachedException.cs

```
1 using System;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Exceptions
6 {
7     public class LinksLimitReachedException<TLinkAddress> : LinksLimitReachedExceptionBase
8     {
9         public LinksLimitReachedException(TLinkAddress limit) : this(FormatMessage(limit)) { }
10
11         public LinksLimitReachedException(string message, Exception innerException) :
12             ↳ base(message, innerException) { }
13
14         public LinksLimitReachedException(string message) : base(message) { }
15
16         public LinksLimitReachedException() : base(DefaultMessage) { }
17
18         private static string FormatMessage(TLinkAddress limit) => $"Достигнут лимит количества
19             ↳ связей в хранилище ({limit}).";
20     }
21 }
```

#### 1.6 ./Platform.Data/Exceptions/LinksLimitReachedExceptionBase.cs

```
1 using System;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Exceptions
6 {
7     public abstract class LinksLimitReachedExceptionBase : Exception
8     {
9         public static readonly string DefaultMessage = "Достигнут лимит количества связей в
10             ↳ хранилище.";
11
12         protected LinksLimitReachedExceptionBase(string message, Exception innerException) :
13             ↳ base(message, innerException) { }
14
15         protected LinksLimitReachedExceptionBase(string message) : base(message) { }
16     }
17 }
```

#### 1.7 ./Platform.Data/Hybrid.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Reflection;
4 using System.Reflection.Emit;
5 using System.Runtime.CompilerServices;
6 using Platform.Exceptions;
7 using Platform.Reflection;
8 using Platform.Converters;
9 using Platform.Numbers;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data
```

```

14 {
15     public struct Hybrid<TLinkAddress> : IEquatable<Hybrid<TLinkAddress>>
16     {
17         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
18             ↳ EqualityComparer<TLinkAddress>.Default;
19         private static readonly UncheckedSignExtendingConverter<TLinkAddress, long>
20             ↳ _addressToInt64Converter = UncheckedSignExtendingConverter<TLinkAddress,
21             ↳ long>.Default;
22         private static readonly UncheckedConverter<TLinkAddress, ulong>
23             ↳ _addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
24         private static readonly UncheckedConverter<ulong, TLinkAddress>
25             ↳ _uint64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
26         private static readonly Func<object, TLinkAddress> _unboxAbsAndConvert =
27             ↳ CompileUnboxAbsAndConvertDelegate();
28         private static readonly Func<object, TLinkAddress> _unboxAbsNegateAndConvert =
29             ↳ CompileUnboxAbsNegateAndConvertDelegate();
30
31         public static readonly ulong HalfOfNumberValuesRange =
32             ↳ _addressToUInt64Converter.Convert(NumericType<TLinkAddress>.MaxValue) / 2;
33         public static readonly TLinkAddress ExternalZero =
34             ↳ _uint64ToAddressConverter.Convert(HalfOfNumberValuesRange + 1UL);
35
36         public readonly TLinkAddress Value;
37
38         public bool IsNothing
39         {
40             [MethodImpl(MethodImplOptions.AggressiveInlining)]
41             get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue == 0;
42         }
43
44         public bool IsInternal
45         {
46             [MethodImpl(MethodImplOptions.AggressiveInlining)]
47             get => SignedValue > 0;
48         }
49
50         public bool IsExternal
51         {
52             [MethodImpl(MethodImplOptions.AggressiveInlining)]
53             get => _equalityComparer.Equals(Value, ExternalZero) || SignedValue < 0;
54         }
55
56         public long SignedValue
57         {
58             [MethodImpl(MethodImplOptions.AggressiveInlining)]
59             get => _addressToInt64Converter.Convert(Value);
60         }
61
62         public long AbsoluteValue
63         {
64             [MethodImpl(MethodImplOptions.AggressiveInlining)]
65             get => _equalityComparer.Equals(Value, ExternalZero) ? 0 :
66                 ↳ Platform.Numbers.Math.Abs(SignedValue);
67         }
68
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         public Hybrid(TLinkAddress value)
71         {
72             Ensure.OnDebug.IsUnsignedInteger<TLinkAddress>();
73             Value = value;
74         }
75
76         [MethodImpl(MethodImplOptions.AggressiveInlining)]
77         public Hybrid(TLinkAddress value, bool isExternal)
78         {
79             if (_equalityComparer.Equals(value, default) && isExternal)
80             {
81                 Value = ExternalZero;
82             }
83             else
84             {
85                 if (isExternal)
86                 {
87                     Value = Math<TLinkAddress>.Negate(value);
88                 }
89                 else
90                 {
91                     Value = value;
92                 }
93             }
94         }
95     }
96 }

```

```

84     }
85
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public Hybrid(object value) => Value =
88         ↪ To.UnsignedAs<TLinkAddress>(Convert.ChangeType(value,
89         ↪ NumericType<TLinkAddress>.SignedVersion));
88
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public Hybrid(object value, bool isExternal)
91     {
92         if (IsDefault(value) && isExternal)
93         {
94             Value = ExternalZero;
95         }
96         else
97         {
98             if (isExternal)
99             {
100                 Value = _unboxAbsNegateAndConvert(value);
101             }
102             else
103             {
104                 Value = _unboxAbsAndConvert(value);
105             }
106         }
107     }
108
109     [MethodImpl(MethodImplOptions.AggressiveInlining)]
110     public static implicit operator Hybrid<TLinkAddress>(TLinkAddress integer) => new
111         ↪ Hybrid<TLinkAddress>(integer);
112
113     [MethodImpl(MethodImplOptions.AggressiveInlining)]
114     public static explicit operator Hybrid<TLinkAddress>(ulong integer) => new
115         ↪ Hybrid<TLinkAddress>(integer);
116
117     [MethodImpl(MethodImplOptions.AggressiveInlining)]
118     public static explicit operator Hybrid<TLinkAddress>(long integer) => new
119         ↪ Hybrid<TLinkAddress>(integer);
120
121     [MethodImpl(MethodImplOptions.AggressiveInlining)]
122     public static explicit operator Hybrid<TLinkAddress>(uint integer) => new
123         ↪ Hybrid<TLinkAddress>(integer);
124
125     [MethodImpl(MethodImplOptions.AggressiveInlining)]
126     public static explicit operator Hybrid<TLinkAddress>(int integer) => new
127         ↪ Hybrid<TLinkAddress>(integer);
128
129     [MethodImpl(MethodImplOptions.AggressiveInlining)]
130     public static explicit operator Hybrid<TLinkAddress>(ushort integer) => new
131         ↪ Hybrid<TLinkAddress>(integer);
132
133     [MethodImpl(MethodImplOptions.AggressiveInlining)]
134     public static explicit operator Hybrid<TLinkAddress>(short integer) => new
135         ↪ Hybrid<TLinkAddress>(integer);
136
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public static explicit operator Hybrid<TLinkAddress>(byte integer) => new
139         ↪ Hybrid<TLinkAddress>(integer);
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static explicit operator Hybrid<TLinkAddress>(sbyte integer) => new
143         ↪ Hybrid<TLinkAddress>(integer);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static implicit operator TLinkAddress(Hybrid<TLinkAddress> hybrid) =>
147         ↪ hybrid.Value;
148
149     [MethodImpl(MethodImplOptions.AggressiveInlining)]
150     public static explicit operator ulong(Hybrid<TLinkAddress> hybrid) =>
151         ↪ CheckedConverter<TLinkAddress, ulong>.Default.Convert(hybrid.Value);
152
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     public static explicit operator long(Hybrid<TLinkAddress> hybrid) =>
155         ↪ hybrid.AbsoluteValue;
156
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     public static explicit operator uint(Hybrid<TLinkAddress> hybrid) =>
159         ↪ CheckedConverter<TLinkAddress, uint>.Default.Convert(hybrid.Value);

```

```

147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 public static explicit operator int(Hybrid<TLinkAddress> hybrid) =>
149     ↪ (int)hybrid.AbsoluteValue;
150
151 [MethodImpl(MethodImplOptions.AggressiveInlining)]
152 public static explicit operator ushort(Hybrid<TLinkAddress> hybrid) =>
153     ↪ CheckedConverter<TLinkAddress, ushort>.Default.Convert(hybrid.Value);
154
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 public static explicit operator short(Hybrid<TLinkAddress> hybrid) =>
157     ↪ (short)hybrid.AbsoluteValue;
158
159 [MethodImpl(MethodImplOptions.AggressiveInlining)]
160 public static explicit operator byte(Hybrid<TLinkAddress> hybrid) =>
161     ↪ CheckedConverter<TLinkAddress, byte>.Default.Convert(hybrid.Value);
162
163 [MethodImpl(MethodImplOptions.AggressiveInlining)]
164 public override string ToString() => IsExternal ? $"<{AbsoluteValue}>" :
165     ↪ Value.ToString();
166
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 public bool Equals(Hybrid<TLinkAddress> other) => _equalityComparer.Equals(Value,
169     ↪ other.Value);
170
171 [MethodImpl(MethodImplOptions.AggressiveInlining)]
172 public override bool Equals(object obj) => obj is Hybrid<TLinkAddress> hybrid ?
173     ↪ Equals(hybrid) : false;
174
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 public override int GetHashCode() => Value.GetHashCode();
177
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 public static bool operator ==(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
180     ↪ left.Equals(right);
181
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 public static bool operator !=(Hybrid<TLinkAddress> left, Hybrid<TLinkAddress> right) =>
184     ↪ !(left == right);
185
186 private static bool IsDefault(object value)
187 {
188     if (value == null)
189     {
190         return true;
191     }
192     var type = value.GetType();
193     return type.IsValueType ? value.Equals(Activator.CreateInstance(type)) : false;
194 }
195
196 private static Func<object, TLinkAddress> CompileUnboxAbsNegateAndConvertDelegate()
197 {
198     return DelegateHelpers.Compile<Func<object, TLinkAddress>>(emitter =>
199     {
200         Ensure.Always.IsUnsignedInteger<TLinkAddress>();
201         emitter.LoadArgument(0);
202         var signedVersion = NumericType<TLinkAddress>.SignedVersion;
203         var signedVersionField =
204             ↪ typeof(NumericType<TLinkAddress>).GetTypeInfo().GetField("SignedVersion",
205             ↪ BindingFlags.Static | BindingFlags.Public);
206         emitter.Emit(OpCodes.Ldsfld, signedVersionField);
207         var changeTypeMethod = typeof(Convert).GetTypeInfo().GetMethod("ChangeType",
208             ↪ Types<object, Type>.Array);
209         emitter.Call(changeTypeMethod);
210         emitter.UnboxValue(signedVersion);
211         var absMethod = typeof(System.Math).GetTypeInfo().GetMethod("Abs", new[] {
212             ↪ signedVersion });
213         emitter.Call(absMethod);
214         var negateMethod = typeof(Platform.Numbers.Math).GetTypeInfo().GetMethod("Negate",
215             ↪ ").MakeGenericMethod(signedVersion);
216         emitter.Call(negateMethod);
217         var unsignedMethod = typeof(To).GetTypeInfo().GetMethod("Unsigned", new[] {
218             ↪ signedVersion });

```

```

208         emitter.Call(unsignedMethod);
209         emitter.Return();
210     });
211 }
212
213 private static Func<object, TLinkAddress> CompileUnboxAbsAndConvertDelegate()
214 {
215     return DelegateHelpers.Compile<Func<object, TLinkAddress>>(emitter =>
216     {
217         Ensure.Always.IsUnsignedInteger<TLinkAddress>();
218         emitter.LoadArgument(0);
219         var signedVersion = NumericType<TLinkAddress>.SignedVersion;
220         var signedVersionField =
221             ↳ typeof(NumericType<TLinkAddress>).GetTypeInfo().GetField("SignedVersion",
222             ↳ BindingFlags.Static | BindingFlags.Public);
223         emitter.Emit(OpCodes.Ldsfld, signedVersionField);
224         var changeTypeMethod = typeof(Convert).GetTypeInfo().GetMethod("ChangeType",
225             ↳ Types<object, Type>.Array);
226         emitter.Call(changeTypeMethod);
227         emitter.UnboxValue(signedVersion);
228         var absMethod = typeof(System.Math).GetTypeInfo().GetMethod("Abs", new[] {
229             ↳ signedVersion });
230         emitter.Call(absMethod);
231         var unsignedMethod = typeof(To).GetTypeInfo().GetMethod("Unsigned", new[] {
232             ↳ signedVersion });
233         emitter.Call(unsignedMethod);
234         emitter.Return();
235     });
236 }
237 }

```

## 1.8 ./Platform.Data/ILinks.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data
7  {
8      /// <summary>
9      /// Представляет интерфейс для работы с данными в формате Links (хранилища взаимосвязей).
10     /// </summary>
11     /// <remarks>
12     /// Этот интерфейс в данный момент не зависит от размера содержимого связи, а значит
13     ↳ подходит как для дуплетов, так и для триплетов и т.п.
14     /// Возможно этот интерфейс подходит даже для Sequences.
15     /// </remarks>
16     public interface ILinks<TLinkAddress, TConstants>
17     where TConstants : LinksConstants<TLinkAddress>
18     {
19         #region Constants
20
21         /// <summary>
22         /// Возвращает набор констант, который необходим для эффективной коммуникации с методами
23         ↳ этого интерфейса.
24         /// Эти константы не меняются с момента создания точки доступа к хранилищу.
25         /// </summary>
26         TConstants Constants { get; }
27
28         #endregion
29
30         #region Read
31
32         /// <summary>
33         /// Подсчитывает и возвращает общее число связей находящихся в хранилище,
34         ↳ соответствующих указанным ограничениям.
35         /// </summary>
36         /// <param name="restriction">Ограничения на содержимое связей.</param>
37         /// <returns>Общее число связей находящихся в хранилище, соответствующих указанным
38         ↳ ограничениям.</returns>
39         TLinkAddress Count(ICollection<TLinkAddress> restriction);
40
41         /// <summary>
42         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
43         ↳ (handler) для каждой подходящей связи.
44         /// </summary>
45         /// <param name="handler">Обработчик каждой подходящей связи.</param>

```

```

41     /// <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>
42     /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
43     TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler, IList<TLinkAddress>
    ↳ restrictions);
44
45     #endregion
46
47     #region Write
48
49     /// <summary>
50     /// Создаёт связь.
51     /// </summary>
52     /// <returns>Индекс созданной связи.</returns>
53     TLinkAddress Create(IList<TLinkAddress> restrictions); // TODO: Возможно всегда нужно
    ↳ принимать restrictions, возможно и возвращать связь нужно целиком.
54
55     /// <summary>
56     /// Обновляет связь с указанными restrictions[Constants.IndexPart] в адресом связи
57     /// на связь с указанным новым содержимым.
58     /// </summary>
59     /// <param name="restrictions">
60     /// Ограничения на содержимое связей.
61     /// Предполагается, что будет указан индекс связи (в restrictions[Constants.IndexPart])
    ↳ и далее за ним будет следовать содержимое связи.
62     /// Каждое ограничение может иметь значения: Constants.Null - 0-я связь, обозначающая
    ↳ ссылку на пустоту,
63     /// Constants.Itself - требование установить ссылку на себя, 1..∞ конкретный индекс
    ↳ другой связи.
64     /// </param>
65     /// <param name="substitution"></param>
66     /// <returns>Индекс обновлённой связи.</returns>
67     TLinkAddress Update(IList<TLinkAddress> restrictions, IList<TLinkAddress> substitution);
    ↳ // TODO: Возможно и возвращать связь нужно целиком.
68
69     /// <summary>Удаляет связь с указанным индексом.</summary>
70     void Delete(IList<TLinkAddress> restrictions); // TODO: Возможно всегда нужно принимать
    ↳ restrictions, а так же возвращать удалённую связь, если удаление было реально
    ↳ выполнено, и Null, если нет.
71
72     #endregion
73 }
74 }

```

## 1.9 ./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     public static class ILinksExtensions
12     {
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public static TLinkAddress Count<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, params TLinkAddress[] restrictions)
15             where TConstants : LinksConstants<TLinkAddress>
16             => links.Count(restrictions);
17
18         /// <summary>
19         /// Возвращает значение, определяющее существует ли связь с указанным индексом в
    ↳ хранилище связей.
20         /// </summary>
21         /// <param name="links">Хранилище связей.</param>
22         /// <param name="link">Индекс проверяемой на существование связи.</param>
23         /// <returns>Значение, определяющее существует ли связь.</returns>
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public static bool Exists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, TLinkAddress link)
26             where TConstants : LinksConstants<TLinkAddress>
27         {
28             var constants = links.Constants;

```

```

29         return constants.IsExternalReference(link) || (constants.IsInternalReference(link)
    ↳ && Comparer<TLinkAddress>.Default.Compare(links.Count(new
    ↳ LinkAddress<TLinkAddress>(link)), default) > 0);
30     }
31
32     /// <param name="links">Хранилище связей.</param>
33     /// <param name="link">Индекс проверяемой на существование связи.</param>
34     /// <remarks>
35     /// TODO: May be move to EnsureExtensions or make it both there and here
36     /// </remarks>
37     [MethodImpl(MethodImplOptions.AggressiveInlining)]
38     public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, TLinkAddress link)
    ↳ where TConstants : LinksConstants<TLinkAddress>
39     {
40
41         if (!links.Exists(link))
42         {
43             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link);
44         }
45     }
46
47     /// <param name="links">Хранилище связей.</param>
48     /// <param name="link">Индекс проверяемой на существование связи.</param>
49     /// <param name="argumentName">Имя аргумента, в который передаётся индекс связи.</param>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, TLinkAddress link, string argumentName)
    ↳ where TConstants : LinksConstants<TLinkAddress>
52     {
53
54         if (!links.Exists(link))
55         {
56             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
57         }
58     }
59
60     /// <summary>
61     /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
62     /// </summary>
63     /// <param name="links">Хранилище связей.</param>
64     /// <param name="handler">Обработчик каждой подходящей связи.</param>
65     /// <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>
66     /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public static TLinkAddress Each<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, Func<IList<TLinkAddress>, TLinkAddress> handler, params
    ↳ TLinkAddress[] restrictions)
    ↳ where TConstants : LinksConstants<TLinkAddress>
69     => links.Each(handler, restrictions);
70
71
72     /// <summary>
73     /// Возвращает части-значения для связи с указанным индексом.
74     /// </summary>
75     /// <param name="links">Хранилище связей.</param>
76     /// <param name="link">Индекс связи.</param>
77     /// <returns>Уникальную связь.</returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     public static IList<TLinkAddress> GetLink<TLinkAddress, TConstants>(this
    ↳ ILinks<TLinkAddress, TConstants> links, TLinkAddress link)
    ↳ where TConstants : LinksConstants<TLinkAddress>
80     {
81
82         var constants = links.Constants;
83         if (constants.IsExternalReference(link))
84         {
85             return new Point<TLinkAddress>(link, constants.TargetPart + 1);
86         }
87         var linkPartsSetter = new Setter<IList<TLinkAddress>,
    ↳ TLinkAddress>(constants.Continue, constants.Break);
88         links.Each(linkPartsSetter.SetAndReturnTrue, link);
89         return linkPartsSetter.Result;
90     }
91
92     #region Points
93

```





```

7     public interface ISynchronizedLinks<TLinkAddress, TLinks, TConstants> :
      ↪     ISynchronized<TLinks>, ILinks<TLinkAddress, TConstants>
8         where TLinks : ILinks<TLinkAddress, TConstants>
9         where TConstants : LinksConstants<TLinkAddress>
10    {
11    }
12 }

```

### 1.11 ./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     public class LinkAddress<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>,
      ↪     IList<TLinkAddress>
11     {
12         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
      ↪     EqualityComparer<TLinkAddress>.Default;
13
14         public TLinkAddress Index
15         {
16             [MethodImpl(MethodImplOptions.AggressiveInlining)]
17             get;
18         }
19
20         public TLinkAddress this[int index]
21         {
22             [MethodImpl(MethodImplOptions.AggressiveInlining)]
23             get
24             {
25                 if (index == 0)
26                 {
27                     return Index;
28                 }
29                 else
30                 {
31                     throw new IndexOutOfRangeException();
32                 }
33             }
34             [MethodImpl(MethodImplOptions.AggressiveInlining)]
35             set => throw new NotSupportedException();
36         }
37
38         public int Count => 1;
39
40         public bool IsReadOnly => true;
41
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public LinkAddress(TLinkAddress index) => Index = index;
44
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public void Add(TLinkAddress item) => throw new NotSupportedException();
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public void Clear() => throw new NotSupportedException();
50
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         public virtual bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index)
      ↪     ? true : false;
53
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
56
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         public IEnumerator<TLinkAddress> GetEnumerator()
59         {
60             yield return Index;
61         }
62
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
      ↪     0 : -1;
65
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();

```

```

68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public bool Remove(TLinkAddress item) => throw new NotSupportedException();
70
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     public void RemoveAt(int index) => throw new NotSupportedException();
73
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     IEnumerator IEnumerable.GetEnumerator()
76     {
77         yield return Index;
78     }
79
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
82         ↪ _equalityComparer.Equals(Index, other.Index);
83
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public static implicit operator TLinkAddress(LinkAddress<TLinkAddress> linkAddress) =>
86         ↪ linkAddress.Index;
87
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public static implicit operator LinkAddress<TLinkAddress>(TLinkAddress linkAddress) =>
90         ↪ new LinkAddress<TLinkAddress>(linkAddress);
91
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public override bool Equals(object obj) => obj is LinkAddress<TLinkAddress> linkAddress
94         ↪ ? Equals(linkAddress) : false;
95
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public override int GetHashCode() => Index.GetHashCode();
98
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override string ToString() => Index.ToString();
101
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    public static bool operator ==(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
104        ↪ right)
105    {
106        if (left == null && right == null)
107        {
108            return true;
109        }
110        if (left == null)
111        {
112            return false;
113        }
114        return left.Equals(right);
115    }
116
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    public static bool operator !=(LinkAddress<TLinkAddress> left, LinkAddress<TLinkAddress>
119        ↪ right) => !(left == right);
120
121 }

```

## 1.12 ./Platform.Data/LinksConstants.cs

```

1  using Platform.Ranges;
2  using Platform.Reflection;
3  using Platform.Converters;
4  using Platform.Numbers;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data
9  {
10     public class LinksConstants<TLinkAddress>
11     {
12         public const int DefaultTargetPart = 2;
13
14         private static readonly TLinkAddress _one = Arithmetic<TLinkAddress>.Increment(default);
15         private static readonly UncheckedConverter<ulong, TLinkAddress>
16             ↪ _uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
17
18         #region Link parts
19
20         /// <summary>Возвращает индекс части, которая отвечает за индекс (адрес, идентификатор)
21         ↪ самой связи.</summary>
22         public int IndexPart { get; }
23     }
24 }

```

```

21
22 /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-начало (первая
   ↳ часть-значение).</summary>
23 public int SourcePart { get; }
24
25 /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-конец
   ↳ (последняя часть-значение).</summary>
26 public int TargetPart { get; }
27
28 #endregion
29
30 #region Flow control
31
32 /// <summary>Возвращает значение, обозначающее продолжение прохода по связям.</summary>
33 /// <remarks>Используется в функции обработчике, который передаётся в функцию
   ↳ Each.</remarks>
34 public TLinkAddress Continue { get; }
35
36 /// <summary>Возвращает значение, обозначающее пропуск в проходе по связям.</summary>
37 public TLinkAddress Skip { get; }
38
39 /// <summary>Возвращает значение, обозначающее остановку прохода по связям.</summary>
40 /// <remarks>Используется в функции обработчике, который передаётся в функцию
   ↳ Each.</remarks>
41 public TLinkAddress Break { get; }
42
43 #endregion
44
45 #region Special symbols
46
47 /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
48 public TLinkAddress Null { get; }
49
50 /// <summary>Возвращает значение, обозначающее любую связь.</summary>
51 /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
   ↳ создавать все варианты последовательностей в функции Create.</remarks>
52 public TLinkAddress Any { get; }
53
54 /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
55 public TLinkAddress Itself { get; }
56
57 #endregion
58
59 #region References
60
61 /// <summary>Возвращает диапазон возможных индексов для внутренних связей (внутренних
   ↳ ссылок).</summary>
62 public Range<TLinkAddress> InternalReferencesRange { get; }
63
64 /// <summary>Возвращает диапазон возможных индексов для внешних связей (внешних
   ↳ ссылок).</summary>
65 public Range<TLinkAddress>? ExternalReferencesRange { get; }
66
67 #endregion
68
69 public LinksConstants(int targetPart, Range<TLinkAddress>
   ↳ possibleInternalReferencesRange, Range<TLinkAddress>?
   ↳ possibleExternalReferencesRange)
70 {
71     IndexPart = 0;
72     SourcePart = 1;
73     TargetPart = targetPart;
74     Null = default;
75     Break = default;
76     var currentInternalReferenceIndex = possibleInternalReferencesRange.Maximum;
77     Continue = currentInternalReferenceIndex;
78     Decrement(ref currentInternalReferenceIndex);
79     Skip = currentInternalReferenceIndex;
80     Decrement(ref currentInternalReferenceIndex);
81     Any = currentInternalReferenceIndex;
82     Decrement(ref currentInternalReferenceIndex);
83     Itself = currentInternalReferenceIndex;
84     Decrement(ref currentInternalReferenceIndex);
85     InternalReferencesRange = (possibleInternalReferencesRange.Minimum,
   ↳ currentInternalReferenceIndex);
86     ExternalReferencesRange = possibleExternalReferencesRange;
87 }
88

```

```

89     public LinksConstants(int targetPart, bool enableExternalReferencesSupport) :
90         ↪ this(targetPart, GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
91         ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
92
93     public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange,
94         ↪ Range<TLinkAddress>? possibleExternalReferencesRange) : this(DefaultTargetPart,
95         ↪ possibleInternalReferencesRange, possibleExternalReferencesRange) { }
96
97     public LinksConstants(bool enableExternalReferencesSupport) :
98         ↪ this(GetDefaultInternalReferencesRange(enableExternalReferencesSupport),
99         ↪ GetDefaultExternalReferencesRange(enableExternalReferencesSupport)) { }
100
101     public LinksConstants(int targetPart, Range<TLinkAddress>
102         ↪ possibleInternalReferencesRange) : this(targetPart, possibleInternalReferencesRange,
103         ↪ null) { }
104
105     public LinksConstants(Range<TLinkAddress> possibleInternalReferencesRange) :
106         ↪ this(DefaultTargetPart, possibleInternalReferencesRange, null) { }
107
108     public LinksConstants() : this(DefaultTargetPart, enableExternalReferencesSupport:
109         ↪ false) { }
110
111     public static Range<TLinkAddress> GetDefaultInternalReferencesRange(bool
112         ↪ enableExternalReferencesSupport)
113     {
114         if (enableExternalReferencesSupport)
115         {
116             return (_one, _uint64ToAddressConverter.Convert(Hybrid<TLinkAddress>.HalfOfNumbe
117                 ↪ rValuesRange));
118         }
119         else
120         {
121             return (_one, NumericType<TLinkAddress>.MaxValue);
122         }
123     }
124
125     public static Range<TLinkAddress>? GetDefaultExternalReferencesRange(bool
126         ↪ enableExternalReferencesSupport)
127     {
128         if (enableExternalReferencesSupport)
129         {
130             return (_uint64ToAddressConverter.Convert(Hybrid<TLinkAddress>.HalfOfNumberValue
131                 ↪ sRange + 1UL),
132                 ↪ NumericType<TLinkAddress>.MaxValue);
133         }
134         else
135         {
136             return null;
137         }
138     }
139
140     private static void Decrement(ref TLinkAddress currentInternalReferenceIndex) =>
141         ↪ currentInternalReferenceIndex = Arithmetic.Decrement(currentInternalReferenceIndex);
142 }

```

### 1.13 ./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      public static class LinksConstantsExtensions
8      {
9          [MethodImpl(MethodImplOptions.AggressiveInlining)]
10         public static bool IsReference<TLinkAddress>(this LinksConstants<TLinkAddress>
11             ↪ linksConstants, TLinkAddress address) => linksConstants.IsInternalReference(address)
12             ↪ || linksConstants.IsExternalReference(address);
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public static bool IsInternalReference<TLinkAddress>(this LinksConstants<TLinkAddress>
16             ↪ linksConstants, TLinkAddress address) =>
17             ↪ linksConstants.InternalReferencesRange.Contains(address);
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

16         public static bool IsExternalReference<TLinkAddress>(<this> LinksConstants<TLinkAddress>
17             ↪ linksConstants, TLinkAddress address) =>
18             ↪ linksConstants.ExternalReferencesRange?.Contains(address) ?? false;
17     }
18 }

```

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

```

1 using Platform.Converters;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Numbers.Raw
6 {
7     public class AddressToRawNumberConverter<TLink> : IConverter<TLink>
8     {
9         public TLink Convert(TLink source) => new Hybrid<TLink>(source, isExternal: true);
10    }
11 }

```

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

```

1 using Platform.Converters;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Numbers.Raw
6 {
7     public class RawNumberToAddressConverter<TLink> : IConverter<TLink>
8     {
9         static private readonly UncheckedConverter<long, TLink> _converter =
10             ↪ UncheckedConverter<long, TLink>.Default;
11
12         public TLink Convert(TLink source) => _converter.Convert(new
13             ↪ Hybrid<TLink>(source).AbsoluteValue);
14    }
15 }

```

#### 1.16 ./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     public class Point<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>, IList<TLinkAddress>
14     {
15         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
16             ↪ EqualityComparer<TLinkAddress>.Default;
17
18         public TLinkAddress Index
19         {
20             [MethodImpl(MethodImplOptions.AggressiveInlining)]
21             get;
22         }
23
24         public int Size
25         {
26             [MethodImpl(MethodImplOptions.AggressiveInlining)]
27             get;
28         }
29
30         public TLinkAddress this[int index]
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get
34             {
35                 if (index < Size)
36                 {
37                     return Index;
38                 }
39                 else
40                 {
41                     throw new IndexOutOfRangeException();
42                 }
43             }
44         }
45     }
46 }

```

```

43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     set => throw new NotSupportedException();
45 }
46
47 public int Count => int.MaxValue;
48
49 public bool IsReadOnly => true;
50
51 [MethodImpl(MethodImplOptions.AggressiveInlining)]
52 public Point(TLinkAddress index, int size)
53 {
54     Index = index;
55     Size = size;
56 }
57
58 [MethodImpl(MethodImplOptions.AggressiveInlining)]
59 public void Add(TLinkAddress item) => throw new NotSupportedException();
60
61 [MethodImpl(MethodImplOptions.AggressiveInlining)]
62 public void Clear() => throw new NotSupportedException();
63
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public virtual bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index)
66     ↪ ? true : false;
67
68 [MethodImpl(MethodImplOptions.AggressiveInlining)]
69 public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;
70
71 [MethodImpl(MethodImplOptions.AggressiveInlining)]
72 public IEnumerator<TLinkAddress> GetEnumerator()
73 {
74     for (int i = 0; i < Size; i++)
75     {
76         yield return Index;
77     }
78 }
79
80 [MethodImpl(MethodImplOptions.AggressiveInlining)]
81 public virtual int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ?
82     ↪ 0 : -1;
83
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
86
87 [MethodImpl(MethodImplOptions.AggressiveInlining)]
88 public bool Remove(TLinkAddress item) => throw new NotSupportedException();
89
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 public void RemoveAt(int index) => throw new NotSupportedException();
92
93 [MethodImpl(MethodImplOptions.AggressiveInlining)]
94 public IEnumerator IEnumerable.GetEnumerator()
95 {
96     for (int i = 0; i < Size; i++)
97     {
98         yield return Index;
99     }
100 }
101
102 [MethodImpl(MethodImplOptions.AggressiveInlining)]
103 public virtual bool Equals(LinkAddress<TLinkAddress> other) => other == null ? false :
104     ↪ _equalityComparer.Equals(Index, other.Index);
105
106 [MethodImpl(MethodImplOptions.AggressiveInlining)]
107 public static implicit operator TLinkAddress(Point<TLinkAddress> linkAddress) =>
108     ↪ linkAddress.Index;
109
110 [MethodImpl(MethodImplOptions.AggressiveInlining)]
111 public override bool Equals(object obj) => obj is Point<TLinkAddress> linkAddress ?
112     ↪ Equals(linkAddress) : false;
113
114 [MethodImpl(MethodImplOptions.AggressiveInlining)]
115 public override int GetHashCode() => Index.GetHashCode();
116
117 [MethodImpl(MethodImplOptions.AggressiveInlining)]
118 public override string ToString() => Index.ToString();
119
120 [MethodImpl(MethodImplOptions.AggressiveInlining)]
121 public static bool operator ==(Point<TLinkAddress> left, Point<TLinkAddress> right)
122 {

```

```

118         if (left == null && right == null)
119         {
120             return true;
121         }
122         if (left == null)
123         {
124             return false;
125         }
126         return left.Equals(right);
127     }
128
129     [MethodImpl(MethodImplOptions.AggressiveInlining)]
130     public static bool operator !=(Point<TLinkAddress> left, Point<TLinkAddress> right) =>
131         ↪ !(left == right);
132
133     [MethodImpl(MethodImplOptions.AggressiveInlining)]
134     public static bool IsFullPoint(params TLinkAddress[] link) =>
135         ↪ IsFullPoint((IList<TLinkAddress>)link);
136
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public static bool IsFullPoint(IList<TLinkAddress> link)
139     {
140         Ensure.Always.ArgumentNotEmpty(link, nameof(link));
141         Ensure.Always.ArgumentInRange(link.Count, new Range<int>(2, int.MaxValue),
142             ↪ nameof(link), "Cannot determine link's pointness using only its identifier.");
143         return IsFullPointUnchecked(link);
144     }
145
146     [MethodImpl(MethodImplOptions.AggressiveInlining)]
147     public static bool IsFullPointUnchecked(IList<TLinkAddress> link)
148     {
149         var result = true;
150         for (var i = 1; result && i < link.Count; i++)
151         {
152             result = _equalityComparer.Equals(link[0], link[i]);
153         }
154         return result;
155     }
156
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     public static bool IsPartialPoint(params TLinkAddress[] link) =>
159         ↪ IsPartialPoint((IList<TLinkAddress>)link);
160
161     [MethodImpl(MethodImplOptions.AggressiveInlining)]
162     public static bool IsPartialPoint(IList<TLinkAddress> link)
163     {
164         Ensure.Always.ArgumentNotEmpty(link, nameof(link));
165         Ensure.Always.ArgumentInRange(link.Count, new Range<int>(2, int.MaxValue),
166             ↪ nameof(link), "Cannot determine link's pointness using only its identifier.");
167         return IsPartialPointUnchecked(link);
168     }
169
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     public static bool IsPartialPointUnchecked(IList<TLinkAddress> link)
172     {
173         var result = false;
174         for (var i = 1; !result && i < link.Count; i++)
175         {
176             result = _equalityComparer.Equals(link[0], link[i]);
177         }
178         return result;
179     }
180 }

```

### 1.17 ./Platform.Data/Sequences/ISequenceAppender.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Sequences
4  {
5      public interface ISequenceAppender<TLinkAddress>
6      {
7          TLinkAddress Append(TLinkAddress sequence, TLinkAddress appendant);
8      }
9  }

```

### 1.18 ./Platform.Data/Sequences/ISequenceWalker.cs

```

1  using System.Collections.Generic;
2

```



```

3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Sequences
6  {
7      public interface ISequenceWalker<TLinkAddress>
8      {
9          IEnumerable<IList<TLinkAddress>> Walk(TLinkAddress sequence);
10     }
11 }

```

#### 1.19 ./Platform.Data/Sequences/SequenceWalker.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.Sequences
8  {
9      /// <remarks>
10     /// Реализованный внутри алгоритм наглядно показывает,
11     /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой
12     /// ↪ себя),
13     /// так как стек можно использовать намного эффективнее при ручном управлении.
14     ///
15     /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
16     /// Решить встраивать ли защиту от заикливания.
17     /// Альтернативой защиты от закливания может быть заранее известное ограничение на
18     /// ↪ погружение вглубь.
19     /// А так же качественное распознавание прохода по циклическому графу.
20     /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
21     /// ↪ стека.
22     /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
23     /// </remarks>
24     public static class SequenceWalker
25     {
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public static void WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
28             ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
29             ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
30         {
31             var stack = new Stack<TLinkAddress>();
32             var element = sequence;
33             if (isElement(element))
34             {
35                 visit(element);
36             }
37             else
38             {
39                 while (true)
40                 {
41                     if (isElement(element))
42                     {
43                         if (stack.Count == 0)
44                         {
45                             break;
46                         }
47                         element = stack.Pop();
48                         var source = getSource(element);
49                         var target = getTarget(element);
50                         if (isElement(source))
51                         {
52                             visit(source);
53                         }
54                         if (isElement(target))
55                         {
56                             visit(target);
57                         }
58                         element = target;
59                     }
60                     else
61                     {
62                         stack.Push(element);
63                         element = getSource(element);
64                     }
65                 }
66             }
67         }
68     }
69 }

```

```

64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public static void WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↳ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↳ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
66 {
67     var stack = new Stack<TLinkAddress>();
68     var element = sequence;
69     if (isElement(element))
70     {
71         visit(element);
72     }
73     else
74     {
75         while (true)
76         {
77             if (isElement(element))
78             {
79                 if (stack.Count == 0)
80                 {
81                     break;
82                 }
83                 element = stack.Pop();
84                 var source = getSource(element);
85                 var target = getTarget(element);
86                 if (isElement(target))
87                 {
88                     visit(target);
89                 }
90                 if (isElement(source))
91                 {
92                     visit(source);
93                 }
94                 element = source;
95             }
96             else
97             {
98                 stack.Push(element);
99                 element = getTarget(element);
100             }
101         }
102     }
103 }
104 }
105 }

```

## 1.20 ./Platform.Data/Sequences/StopableSequenceWalker.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.Sequences
8 {
9     /// <remarks>
10     /// Реализованный внутри алгоритм наглядно показывает,
11     /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой
    ↳ себя),
12     /// так как стек можно использовать намного эффективнее при ручном управлении.
13     ///
14     /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
15     /// Решить встраивать ли защиту от заикливания.
16     /// Альтернативой защиты от закливания может быть заранее известное ограничение на
    ↳ погружение вглубь.
17     /// А так же качественное распознавание прохода по циклическому графу.
18     /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
    ↳ стека.
19     /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
20     /// </remarks>
21     public static class StopableSequenceWalker
22     {
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↳ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↳ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> enter, Action<TLinkAddress>
    ↳ exit, Func<TLinkAddress, bool> canEnter, Func<TLinkAddress, bool> visit)
25     {
26         var exited = 0;
27         var stack = new Stack<TLinkAddress>();

```

```

28     var element = sequence;
29     if (isElement(element))
30     {
31         return visit(element);
32     }
33     while (true)
34     {
35         if (isElement(element))
36         {
37             if (stack.Count == 0)
38             {
39                 return true;
40             }
41             element = stack.Pop();
42             exit(element);
43             exited++;
44             var source = getSource(element);
45             var target = getTarget(element);
46             if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
                ↪ !visit(source))
47             {
48                 return false;
49             }
50             if ((isElement(target) || !canEnter(target)) && !visit(target))
51             {
52                 return false;
53             }
54             element = target;
55         }
56         else
57         {
58             if (canEnter(element))
59             {
60                 enter(element);
61                 exited = 0;
62                 stack.Push(element);
63                 element = getSource(element);
64             }
65             else
66             {
67                 if (stack.Count == 0)
68                 {
69                     return true;
70                 }
71                 element = stack.Pop();
72                 exit(element);
73                 exited++;
74                 var source = getSource(element);
75                 var target = getTarget(element);
76                 if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
                    ↪ !visit(source))
77                 {
78                     return false;
79                 }
80                 if ((isElement(target) || !canEnter(target)) && !visit(target))
81                 {
82                     return false;
83                 }
84                 element = target;
85             }
86         }
87     }
88 }
89
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
92 {
93     var stack = new Stack<TLinkAddress>();
94     var element = sequence;
95     if (isElement(element))
96     {
97         return visit(element);
98     }
99     while (true)
100     {
101         if (isElement(element))
102         {

```

```

103         if (stack.Count == 0)
104         {
105             return true;
106         }
107         element = stack.Pop();
108         var source = getSource(element);
109         var target = getTarget(element);
110         if (isElement(source) && !visit(source))
111         {
112             return false;
113         }
114         if (isElement(target) && !visit(target))
115         {
116             return false;
117         }
118         element = target;
119     }
120     else
121     {
122         stack.Push(element);
123         element = getSource(element);
124     }
125 }
126 }
127
128 [MethodImpl(MethodImplOptions.AggressiveInlining)]
129 public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
130 {
131     var stack = new Stack<TLinkAddress>();
132     var element = sequence;
133     if (isElement(element))
134     {
135         return visit(element);
136     }
137     while (true)
138     {
139         if (isElement(element))
140         {
141             if (stack.Count == 0)
142             {
143                 return true;
144             }
145             element = stack.Pop();
146             var source = getSource(element);
147             var target = getTarget(element);
148             if (isElement(target) && !visit(target))
149             {
150                 return false;
151             }
152             if (isElement(source) && !visit(source))
153             {
154                 return false;
155             }
156             element = source;
157         }
158         else
159         {
160             stack.Push(element);
161             element = getTarget(element);
162         }
163     }
164 }
165 }
166 }

```

## 1.21 ./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         IList<IList<TLinkAddress>>> Trigger(IList<TLinkAddress> condition,
13         ↪ IList<TLinkAddress> substitution);
14     }
15     /// <remarks>Minimal sufficient universal Links API (for step by step operations).</remarks>
16     public partial interface IUniLinks<TLinkAddress>
17     {
18         /// <returns>
19         /// TLinkAddress that represents True (was finished fully) or TLinkAddress that
20         ↪ represents False (was stopped).
21         /// This is done to assure ability to push up stop signal through recursion stack.
22         /// </returns>
23         /// <remarks>
24         /// { 0, 0, 0 } => { itself, itself, itself } // create
25         /// { 1, any, any } => { itself, any, 3 } // update
26         /// { 3, any, any } => { 0, 0, 0 } // delete
27         /// </remarks>
28         TLinkAddress Trigger(IList<TLinkAddress> patternOrCondition, Func<IList<TLinkAddress>,
29         ↪ TLinkAddress> matchHandler,
30         ↪ IList<TLinkAddress> substitution, Func<IList<TLinkAddress>,
31         ↪ IList<TLinkAddress>, TLinkAddress> substitutionHandler);
32
33         TLinkAddress Trigger(IList<TLinkAddress> restriction, Func<IList<TLinkAddress>,
34         ↪ IList<TLinkAddress>, TLinkAddress> matchedHandler,
35         ↪ IList<TLinkAddress> substitution, Func<IList<TLinkAddress>, IList<TLinkAddress>,
36         ↪ TLinkAddress> substitutedHandler);
37     }
38
39     /// <remarks>Extended with small optimization.</remarks>
40     public partial interface IUniLinks<TLinkAddress>
41     {
42         /// <remarks>
43         /// Something simple should be simple and optimized.
44         /// </remarks>
45         TLinkAddress Count(IList<TLinkAddress> restrictions);
46     }
47 }

```

## 1.22 ./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         TLinkAddress Read(int partType, TLinkAddress link);
15         TLinkAddress Read(Func<TLinkAddress, bool> handler, IList<TLinkAddress> pattern);
16         TLinkAddress Create(IList<TLinkAddress> parts);
17         TLinkAddress Update(IList<TLinkAddress> before, IList<TLinkAddress> after);
18         void Delete(IList<TLinkAddress> parts);
19     }
20 }

```

## 1.23 ./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         TLinkAddress Get(int partType, TLinkAddress link);
15         TLinkAddress Get(Func<TLinkAddress, bool> handler, IList<TLinkAddress> pattern);
16         TLinkAddress Set(IList<TLinkAddress> before, IList<TLinkAddress> after);
17     }
18 }

```

## 1.24 ./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>
52         TLinkAddress In(IList<TLinkAddress> before, IList<TLinkAddress> after);
53     }
54 }

```

## 1.25 ./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         ///
23         ///
24         ///
25         ///
26         ///
27         ///
28         ///
29         ///
30         ///
31         ///
32         ///
33         ///
34         ///
35         ///
36         ///
37         ///
38         ///
39         ///
40         ///
41         ///
42         ///
43         ///
44         ///
45         ///
46         ///
47         ///
48         ///
49         ///
50         ///
51         ///
52         ///
53         ///
54         ///
55         ///
56         ///
57         ///
58         ///
59         ///
60         ///
61         ///
62         ///
63         ///
64         ///
65         ///
66         ///
67         ///
68         ///
69         ///
70         ///
71         ///
72         ///
73         ///
74         ///
75         ///
76         ///
77         ///
78         ///
79         ///
80         ///
81         ///
82         ///
83         ///
84         ///
85         ///
86         ///
87         ///
88         ///
89         ///
90         ///
91         ///
92         ///
93         ///
94         ///
95         ///
96         ///
97         ///
98         ///
99         ///
100        ///
101        ///
102        ///
103        ///
104        ///
105        ///
106        ///
107        ///
108        ///
109        ///
110        ///
111        ///
112        ///
113        ///
114        ///
115        ///
116        ///
117        ///
118        ///
119        ///
120        ///
121        ///
122        ///
123        ///
124        ///
125        ///
126        ///
127        ///
128        ///
129        ///
130        ///
131        ///
132        ///
133        ///
134        ///
135        ///
136        ///
137        ///
138        ///
139        ///
140        ///
141        ///
142        ///
143        ///
144        ///
145        ///
146        ///
147        ///
148        ///
149        ///
150        ///
151        ///
152        ///
153        ///
154        ///
155        ///
156        ///
157        ///
158        ///
159        ///
160        ///
161        ///
162        ///
163        ///
164        ///
165        ///
166        ///
167        ///
168        ///
169        ///
170        ///
171        ///
172        ///
173        ///
174        ///
175        ///
176        ///
177        ///
178        ///
179        ///
180        ///
181        ///
182        ///
183        ///
184        ///
185        ///
186        ///
187        ///
188        ///
189        ///
190        ///
191        ///
192        ///
193        ///
194        ///
195        ///
196        ///
197        ///
198        ///
199        ///
200        ///
201        ///
202        ///
203        ///
204        ///
205        ///
206        ///
207        ///
208        ///
209        ///
210        ///
211        ///
212        ///
213        ///
214        ///
215        ///
216        ///
217        ///
218        ///
219        ///
220        ///
221        ///
222        ///
223        ///
224        ///
225        ///
226        ///
227        ///
228        ///
229        ///
230        ///
231        ///
232        ///
233        ///
234        ///
235        ///
236        ///
237        ///
238        ///
239        ///
240        ///
241        ///
242        ///
243        ///
244        ///
245        ///
246        ///
247        ///
248        ///
249        ///
250        ///
251        ///
252        ///
253        ///
254        ///
255        ///
256        ///
257        ///
258        ///
259        ///
260        ///
261        ///
262        ///
263        ///
264        ///
265        ///
266        ///
267        ///
268        ///
269        ///
270        ///
271        ///
272        ///
273        ///
274        ///
275        ///
276        ///
277        ///
278        ///
279        ///
280        ///
281        ///
282        ///
283        ///
284        ///
285        ///
286        ///
287        ///
288        ///
289        ///
290        ///
291        ///
292        ///
293        ///
294        ///
295        ///
296        ///
297        ///
298        ///
299        ///
300        ///
301        ///
302        ///
303        ///
304        ///
305        ///
306        ///
307        ///
308        ///
309        ///
310        ///
311        ///
312        ///
313        ///
314        ///
315        ///
316        ///
317        ///
318        ///
319        ///
320        ///
321        ///
322        ///
323        ///
324        ///
325        ///
326        ///
327        ///
328        ///
329        ///
330        ///
331        ///
332        ///
333        ///
334        ///
335        ///
336        ///
337        ///
338        ///
339        ///
340        ///
341        ///
342        ///
343        ///
344        ///
345        ///
346        ///
347        ///
348        ///
349        ///
350        ///
351        ///
352        ///
353        ///
354        ///
355        ///
356        ///
357        ///
358        ///
359        ///
360        ///
361        ///
362        ///
363        ///
364        ///
365        ///
366        ///
367        ///
368        ///
369        ///
370        ///
371        ///
372        ///
373        ///
374        ///
375        ///
376        ///
377        ///
378        ///
379        ///
380        ///
381        ///
382        ///
383        ///
384        ///
385        ///
386        ///
387        ///
388        ///
389        ///
390        ///
391        ///
392        ///
393        ///
394        ///
395        ///
396        ///
397        ///
398        ///
399        ///
400        ///
401        ///
402        ///
403        ///
404        ///
405        ///
406        ///
407        ///
408        ///
409        ///
410        ///
411        ///
412        ///
413        ///
414        ///
415        ///
416        ///
417        ///
418        ///
419        ///
420        ///
421        ///
422        ///
423        ///
424        ///
425        ///
426        ///
427        ///
428        ///
429        ///
430        ///
431        ///
432        ///
433        ///
434        ///
435        ///
436        ///
437        ///
438        ///
439        ///
440        ///
441        ///
442        ///
443        ///
444        ///
445        ///
446        ///
447        ///
448        ///
449        ///
450        ///
451        ///
452        ///
453        ///
454        ///
455        ///
456        ///
457        ///
458        ///
459        ///
460        ///
461        ///
462        ///
463        ///
464        ///
465        ///
466        ///
467        ///
468        ///
469        ///
470        ///
471        ///
472        ///
473        ///
474        ///
475        ///
476        ///
477        ///
478        ///
479        ///
480        ///
481        ///
482        ///
483        ///
484        ///
485        ///
486        ///
487        ///
488        ///
489        ///
490        ///
491        ///
492        ///
493        ///
494        ///
495        ///
496        ///
497        ///
498        ///
499        ///
500        ///
501        ///
502        ///
503        ///
504        ///
505        ///
506        ///
507        ///
508        ///
509        ///
510        ///
511        ///
512        ///
513        ///
514        ///
515        ///
516        ///
517        ///
518        ///
519        ///
520        ///
521        ///
522        ///
523        ///
524        ///
525        ///
526        ///
527        ///
528        ///
529        ///
530        ///
531        ///
532        ///
533        ///
534        ///
535        ///
536        ///
537        ///
538        ///
539        ///
540        ///
541        ///
542        ///
543        ///
544        ///
545        ///
546        ///
547        ///
548        ///
549        ///
550        ///
551        ///
552        ///
553        ///
554        ///
555        ///
556        ///
557        ///
558        ///
559        ///
560        ///
561        ///
562        ///
563        ///
564        ///
565        ///
566        ///
567        ///
568        ///
569        ///
570        ///
571        ///
572        ///
573        ///
574        ///
575        ///
576        ///
577        ///
578        ///
579        ///
580        ///
581        ///
582        ///
583        ///
584        ///
585        ///
586        ///
587        ///
588        ///
589        ///
590        ///
591        ///
592        ///
593        ///
594        ///
595        ///
596        ///
597        ///
598        ///
599        ///
600        ///
601        ///
602        ///
603        ///
604        ///
605        ///
606        ///
607        ///
608        ///
609        ///
610        ///
611        ///
612        ///
613        ///
614        ///
615        ///
616        ///
617        ///
618        ///
619        ///
620        ///
621        ///
622        ///
623        ///
624        ///
625        ///
626        ///
627        ///
628        ///
629        ///
630        ///
631        ///
632        ///
633        ///
634        ///
635        ///
636        ///
637        ///
638        ///
639        ///
640        ///
641        ///
642        ///
643        ///
644        ///
645        ///
646        ///
647        ///
648        ///
649        ///
650        ///
651        ///
652        ///
653        ///
654        ///
655        ///
656        ///
657        ///
658        ///
659        ///
660        ///
661        ///
662        ///
663        ///
664        ///
665        ///
666        ///
667        ///
668        ///
669        ///
670        ///
671        ///
672        ///
673        ///
674        ///
675        ///
676        ///
677        ///
678        ///
679        ///
680        ///
681        ///
682        ///
683        ///
684        ///
685        ///
686        ///
687        ///
688        ///
689        ///
690        ///
691        ///
692        ///
693        ///
694        ///
695        ///
696        ///
697        ///
698        ///
699        ///
700        ///
701        ///
702        ///
703        ///
704        ///
705        ///
706        ///
707        ///
708        ///
709        ///
710        ///
711        ///
712        ///
713        ///
714        ///
715        ///
716        ///
717        ///
718        ///
719        ///
720        ///
721        ///
722        ///
723        ///
724        ///
725        ///
726        ///
727        ///
728        ///
729        ///
730        ///
731        ///
732        ///
733        ///
734        ///
735        ///
736        ///
737        ///
738        ///
739        ///
740        ///
741        ///
742        ///
743        ///
744        ///
745        ///
746        ///
747        ///
748        ///
749        ///
750        ///
751        ///
752        ///
753        ///
754        ///
755        ///
756        ///
757        ///
758        ///
759        ///
760        ///
761        ///
762        ///
763        ///
764        ///
765        ///
766        ///
767        ///
768        ///
769        ///
770        ///
771        ///
772        ///
773        ///
774        ///
775        ///
776        ///
777        ///
778        ///
779        ///
780        ///
781        ///
782        ///
783        ///
784        ///
785        ///
786        ///
787        ///
788        ///
789        ///
790        ///
791        ///
792        ///
793        ///
794        ///
795        ///
796        ///
797        ///
798        ///
799        ///
800        ///
801        ///
802        ///
803        ///
804        ///
805        ///
806        ///
807        ///
808        ///
809        ///
810        ///
811        ///
812        ///
813        ///
814        ///
815        ///
816        ///
817        ///
818        ///
819        ///
820        ///
821        ///
822        ///
823        ///
824        ///
825        ///
826        ///
827        ///
828        ///
829        ///
830        ///
831        ///
832        ///
833        ///
834        ///
835        ///
836        ///
837        ///
838        ///
839        ///
840        ///
841        ///
842        ///
843        ///
844        ///
845        ///
846        ///
847        ///
848        ///
849        ///
850        ///
851        ///
852        ///
853        ///
854        ///
855        ///
856        ///
857        ///
858        ///
859        ///
860        ///
861        ///
862        ///
863        ///
864        ///
865        ///
866        ///
867        ///
868        ///
869        ///
870        ///
871        ///
872        ///
873        ///
874        ///
875        ///
876        ///
877        ///
878        ///
879        ///
880        ///
881        ///
882        ///
883        ///
884        ///
885        ///
886        ///
887        ///
888        ///
889        ///
890        ///
891        ///
892        ///
893        ///
894        ///
895        ///
896        ///
897        ///
898        ///
899        ///
900        ///
901        ///
902        ///
903        ///
904        ///
905        ///
906        ///
907        ///
908        ///
909        ///
910        ///
911        ///
912        ///
913        ///
914        ///
915        ///
916        ///
917        ///
918        ///
919        ///
920        ///
921        ///
922        ///
923        ///
924        ///
925        ///
926        ///
927        ///
928        ///
929        ///
930        ///
931        ///
932        ///
933        ///
934        ///
935        ///
936        ///
937        ///
938        ///
939        ///
940        ///
941        ///
942        ///
943        ///
944        ///
945        ///
946        ///
947        ///
948        ///
949        ///
950        ///
951        ///
952        ///
953        ///
954        ///
955        ///
956        ///
957        ///
958        ///
959        ///
960        ///
961        ///
962        ///
963        ///
964        ///
965        ///
966        ///
967        ///
968        ///
969        ///
970        ///
971        ///
972        ///
973        ///
974        ///
975        ///
976        ///
977        ///
978        ///
979        ///
980        ///
981        ///
982        ///
983        ///
984        ///
985        ///
986        ///
987        ///
988        ///
989        ///
990        ///
991        ///
992        ///
993        ///
994        ///
995        ///
996        ///
997        ///
998        ///
999        ///
1000       ///
1001       ///
1002       ///
1003       ///
1004       ///
1005       ///
1006       ///
1007       ///
1008       ///
1009       ///
1010       ///
1011       ///
1012       ///
1013       ///
1014       ///
1015       ///
1016       ///
1017       ///
1018       ///
1019       ///
1020       ///
1021       ///
1022       ///
1023       ///
1024       ///
1025       ///
1026       ///
1027       ///
1028       ///
1029       ///
1030       ///
1031       ///
1032       ///
1033       ///
1034       ///
1035       ///
1036       ///
1037       ///
1038       ///
1039       ///
1040       ///
1041       ///
1042       ///
1043       ///
1044       ///
1045       ///
1046       ///
1047       ///
1048       ///
1049       ///
1050       ///
1051       ///
1052       ///
1053       ///
1054       ///
1055       ///
1056       ///
1057       ///
1058       ///
1059       ///
1060       ///
1061       ///
1062       ///
1063       ///
1064       ///
1065       ///
1066       ///
1067       ///
1068       ///
1069       ///
1070       ///
1071       ///
1072       ///
1073       ///
1074       ///
1075       ///
1076       ///
1077       ///
1078       ///
1079       ///
1080       ///
1081       ///
1082       ///
1083       ///
1084       ///
1085       ///
1086       ///
1087       ///
1088       ///
1089       ///
1090       ///
1091       ///
1092       ///
1093       ///
1094       ///
1095       ///
1096       ///
1097       ///
1098       ///
1099       ///
1100       ///
1101       ///
1102       ///
1103       ///
1104       ///
1105       ///
1106       ///
1107       ///
1108       ///
1109       ///
1110       ///
1111       ///
1112       ///
1113       ///
1114       ///
1115       ///
1116       ///
1117       ///
1118       ///
1119       ///
1120       ///
1121       ///
1122       ///
1123       ///
1124       ///
1125       ///
1126       ///
1127       ///
1128       ///
1129       ///
1130       ///
1131       ///
1132       ///
1133       ///
1134       ///
1135       ///
1136       ///
1137       ///
1138       ///
1139       ///
1140       ///
1141       ///
1142       ///
1143       ///
1144       ///
1145       ///
1146       ///
1147       ///
1148       ///
1149       ///
1150       ///
1151       ///
1152       ///
1153       ///
1154       ///
1155       ///
1156       ///
1157       ///
1158       ///
1159       ///
1160       ///
1161       ///
1162       ///
1163       ///
1164       ///
1165       ///
1166       ///
1167       ///
1168       ///
1169       ///
1170       ///
1171       ///
1172       ///
1173       ///
1174       ///
1175       ///
1176       ///
1177       ///
1178       ///
1179       ///
1180       ///
1181       ///
1182       ///
1183       ///
1184       ///
1185       ///
1186       ///
1187       ///
1188       ///
1189       ///
1190       ///
1191       ///
1192       ///
1193       ///
1194       ///
1195       ///
1196       ///
1197       ///
1198       ///
1199       ///
1200       ///
1201       ///
1202       ///
1203       ///
1204       ///
1205       ///
1206       ///
1207       ///
1208       ///
1209       ///
1210       ///
1211       ///
1212       ///
1213       ///
1214       ///
1215       ///
1216       ///
1217       ///
1218       ///
1219       ///
1220       ///
1221       ///
1222       ///
1223       ///
1224       ///
1225       ///
1226       ///
1227       ///
1228       ///
1229       ///
1230       ///
1231       ///
1232       ///
1233       ///
1234       ///
1235       ///
1236       ///
1237       ///
1238       ///
1239       ///
1240       ///
1241       ///
1242       ///
1243       ///
1244       ///
1245       ///
1246       ///
1247       ///
1248       ///
1249       ///
1250       ///
1251       ///
1252       ///
1253       ///
1254       ///
1255       ///
1256       ///
1257       ///
1258       ///
1259       ///
1260       ///
1261       ///
1262       ///
1263       ///
1264       ///
1265       ///
1266       ///
1267       ///
1268       ///
1269       ///
1270       ///
1271       ///
1272       ///
1273       ///
1274       ///
1275       ///
1276       ///
1277       ///
1278       ///
1279       ///
1280       ///
1281       ///
1282       ///
1283       ///
1284       ///
1285       ///
1286       ///
1287       ///
1288       ///
1289       ///
1290       ///
1291       ///
1292       ///
1293       ///
1294       ///
1295       ///
1296       ///
1297       ///
1298       ///
1299       ///
1300       ///
1301       ///
1302       ///
1303       ///
1304       ///
1305       ///
1306       ///
1307       ///
1308       ///
1309       ///
1310       ///
1311       ///
1312       ///
1313       ///
1314       ///
1315       ///
1316       ///
1317       ///
1318       ///
1319       ///
1320       ///
1321       ///
1322       ///
1323       ///
1324       ///
1325       ///
1326       ///
1327       ///
1328       ///
1329       ///
1330       ///
1331       ///
1332       ///
1333       ///
1334       ///
1335       ///
1336       ///
1337       ///
1338       ///
1339       ///
1340       ///
1341       ///
1342       ///
1343       ///
1344       ///
1345       ///
1346       ///
1347       ///
1348       ///
1349       ///
1350       ///
1351       ///
1352       ///
1353       ///
1354       ///
1355       ///
1356       ///
1357       ///
1358       ///
1359       ///
1360       ///
1361       ///
1362       ///
1363       ///
1364       ///
1365       ///
1366       ///
1367       ///
1368       ///
1369       ///
1370       ///
1371       ///
1372       ///
1373       ///
1374       ///
1375       ///
1376       ///
1377       ///
1378       ///
1379       ///
1380       ///
1381       ///
1382       ///
1383       ///
1384       ///
1385       ///
1386       ///
1387       ///
1388       ///
1389       ///
1390       ///
1391       ///
1392       ///
1393       ///
1394       ///
1395       ///
1396       ///
1397       ///
1398       ///
1399       ///
1400       ///
1401       ///
1402       ///
1403       ///
1404       ///
1405       ///
1406       ///
1407       ///
1408       ///
1409       ///
1410       ///
1411       ///
1412       ///
1413       ///
1414       ///
1415       ///
1416       ///
1417       ///
1418       ///
1419       ///
1420       ///
1421       ///
1422       ///
1423       ///
1424       ///
1425       ///
1426       ///
1427       ///
1428       ///
1429       ///
1430       ///
1431       ///
1432       ///
1433       ///
1434       ///
1435       ///
1436       ///
1437       ///
1438       ///
1439       ///
1440       ///
1441       ///
1442       ///
1443       ///
1444       ///
1445       ///
1446       ///
1447       ///
1448       ///
1449       ///
1450       ///
1451       ///
1452       ///
1453       ///
1454       ///
1455       ///
1456       ///
1457       ///
1458       ///
1459       ///
1460       ///
1461       ///
1462       ///
1463       ///
1464       ///
1465       ///
1466       ///
1467       ///
1468       ///
1469       ///
1470       ///
1471       ///
1472       ///
1473       ///
1474       ///
1475       ///
1476       ///
1477       ///
1478       ///
1479       ///
1480       ///
1481       ///
1482       ///
1483       ///
1484       ///
1485       ///
1486       ///
1487       ///
1488       ///
1489       ///
1490       ///
1491       ///
1492       ///
1493       ///
1494       ///
1495       ///
1496       ///
1497       ///
1498       ///
1499       ///
1500       ///
1501       ///
1502       ///
1503       ///
1504       ///
1505       ///
1506       ///
1507       ///
1508       ///
1509       ///
1510       ///
1511       ///
1512       ///
1513       ///
1514       ///
1515       ///
1516       ///
1517       ///
1518       ///
1519       ///
1520       ///
1521       ///
1522       ///
1523       ///
1524       ///
1525       ///
1526       ///
1527       ///
1528       ///
1529       ///
1530       ///
1531       ///
1532       ///
1533       ///
1534       ///
1535       ///
1536       ///
1537       ///
1538       ///
1539       ///
1540       ///
1541       ///
1542       ///
1543       ///
1544       ///
1545       ///
1546       ///
1547       ///
1548       ///
1549       ///
1550       ///
1551       ///
1552       ///
1553       ///
1554       ///
1555       ///
1556       ///
1557       ///
1558       ///
1559       ///
1560       ///
1561       ///
1562       ///
1563       ///
1564       ///
1565       ///
1566       ///
1567       ///
1568       ///
1569       ///
1570       ///
1571       ///
1572       ///
1573       ///
1574       ///
1575       ///
1576       ///
1577       ///
1578       ///
1579       ///
1580       ///
1581       ///
1582       ///
1583       ///
1584       ///
1585       ///
1586       ///
1587       ///
1588       ///
1589       ///
1590       ///
1591       ///
1592       ///
1593       ///
1594       ///
1595       ///
1596       ///
1597       ///
1598       ///
1599       ///
1600       ///
1601       ///
1602       ///
1603       ///
1604       ///
1605       ///
1606       ///
1607       ///
1608       ///
1609       ///
1610       ///
1611       ///
1612       ///
1613       ///
1614       ///
1615       ///
1616       ///
1617       ///
1618       ///
1619       ///
1620       ///
1621       ///
1622       ///
1623       ///
1624       ///
1625       ///
1626       ///
1627       ///
1628       ///
1629       ///
1630       ///
1631       ///
1632       ///
1633       ///
1634       ///
1635       ///
1636       ///
1637       ///
1638       ///
1639       ///
1640       ///
1641       ///
1642       ///
1643       ///
1644       ///
1645       ///
1646       ///
1647       ///
1648       ///
1649       ///
1650       ///
1651       ///
1652       ///
1653       ///
1654       ///
1655       ///
1656       ///
1657       ///
1658       ///
1659       ///
1660       ///
1661       ///
1662       ///
1663       ///
1664       ///
1665       ///
1666       ///
1667       ///
1668       ///
1669       ///
1670       ///
1671       ///
1672       ///
1673       ///
1674       ///
1675       ///
1676       ///
1677       ///
1678       ///
1679       ///
1680       ///
1681       ///
1682       ///
1683       ///
1684       ///
1685       ///
1686       ///
1687       ///
1688       ///
1689       ///
1690       ///
1691       ///
1692       ///
1693       ///
1694       ///
1695       ///
1696       ///
1697       ///
1698       ///
1699       ///
1700       ///
1701       ///
1702       ///
1703       ///
1704       ///
1705       ///
1706       ///
1707       ///
1708       ///
1709       ///
1710       ///
1711       ///
1712       ///
1713       ///
1714       ///
1715       ///
1716       ///
1717       ///
1718       ///
1719       ///
1720       ///
1721       ///
1722       ///
1723       ///
1724       ///
1725       ///
1726       ///
1727       ///
1728       ///
1729       ///
1730       ///
1731       ///
1732       ///
1733       ///
1734       ///
1735       ///
1736       ///
1737       ///
1738       ///
1739       ///
1740       ///
1741       ///
1742       ///
1743       ///
1744       ///
1745       ///
1746       ///
1747       ///
1748       ///
1749       ///
1750       ///
1751       ///
1752       ///
1753       ///
1754       ///
1755       ///
1756       ///
1757       ///
1758       ///
1759       ///
1760       ///
1761       ///
1762       ///
1763       ///
1764       ///
1765       ///
1766       ///
1767       ///
1768       ///
1769       ///
1770
```

```

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

```

## 1.26 ./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         TLinkAddress Read(int partType, TLinkAddress link);
15         bool Read(Func<TLinkAddress, bool> handler, IList<TLinkAddress> pattern);
16         TLinkAddress Write(IList<TLinkAddress> before, IList<TLinkAddress> after);
17     }
18 }

```

## 1.27 ./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      public static class LinksConstantsTests
9      {
10         [Fact]
11         public static void ExternalReferencesTest()
12         {
13             TestExternalReferences<ulong, long>();
14             TestExternalReferences<uint, int>();
15             TestExternalReferences<ushort, short>();
16             TestExternalReferences<byte, sbyte>();
17         }
18
19         private static void TestExternalReferences<TUnsigned, TSigned>()
20         {
21             var unsingedOne = Arithmetic.Increment(default(TUnsigned));
22             var converter = UncheckedConverter<TSigned, TUnsigned>.Default;
23             var half = converter.Convert(NumericType<TSigned>.MaxValue);
24             LinksConstants<TUnsigned> constants = new LinksConstants<TUnsigned>((unsingedOne,
25                 ↪ half), (Arithmetic.Add(half, unsingedOne), NumericType<TUnsigned>.MaxValue));
26
27             var minimum = new Hybrid<TUnsigned>(default, isExternal: true);
28             var maximum = new Hybrid<TUnsigned>(half, isExternal: true);
29
30             Assert.True(constants.IsExternalReference(minimum));
31             Assert.True(minimum.IsExternal);
32             Assert.False(minimum.IsInternal);
33             Assert.True(constants.IsExternalReference(maximum));
34             Assert.True(maximum.IsExternal);
35             Assert.False(maximum.IsInternal);
36         }
37     }
38 }

```

## Index

- ./Platform.Data.Tests/LinksConstantsTests.cs, 23
- ./Platform.Data/Exceptions/ArgumentLinkDoesNotExistsException.cs, 1
- ./Platform.Data/Exceptions/ArgumentLinkHasDependenciesException.cs, 1
- ./Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsException.cs, 1
- ./Platform.Data/Exceptions/LinkWithSameValueAlreadyExistsExceptionBase.cs, 2
- ./Platform.Data/Exceptions/LinksLimitReachedException.cs, 2
- ./Platform.Data/Exceptions/LinksLimitReachedExceptionBase.cs, 2
- ./Platform.Data/Hybrid.cs, 2
- ./Platform.Data/ILinks.cs, 6
- ./Platform.Data/ILinksExtensions.cs, 7
- ./Platform.Data/ISynchronizedLinks.cs, 9
- ./Platform.Data/LinkAddress.cs, 10
- ./Platform.Data/LinksConstants.cs, 11
- ./Platform.Data/LinksConstantsExtensions.cs, 13
- ./Platform.Data/Numbers/Raw/AddressToRawNumberConverter.cs, 14
- ./Platform.Data/Numbers/Raw/RawNumberToAddressConverter.cs, 14
- ./Platform.Data/Point.cs, 14
- ./Platform.Data/Sequences/ISequenceAppender.cs, 16
- ./Platform.Data/Sequences/ISequenceWalker.cs, 16
- ./Platform.Data/Sequences/SequenceWalker.cs, 17
- ./Platform.Data/Sequences/StopableSequenceWalker.cs, 18
- ./Platform.Data/Universal/IUniLinks.cs, 20
- ./Platform.Data/Universal/IUniLinksCRUD.cs, 21
- ./Platform.Data/Universal/IUniLinksGS.cs, 21
- ./Platform.Data/Universal/IUniLinksIO.cs, 22
- ./Platform.Data/Universal/IUniLinksIOWithExtensions.cs, 22
- ./Platform.Data/Universal/IUniLinksRW.cs, 23