

# LinksPlatform's Platform.Data Class Library

## ./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 paramName) :
10             ↪ base(FormatMessage(link, paramName), paramName) { }
11          public ArgumentLinkDoesNotExistsException(TLinkAddress link) : base(FormatMessage(link))
12             ↪ { }
13          private static string FormatMessage(TLinkAddress link, string paramName) => $"Связь
14             ↪ [{link}] переданная в аргумент [{paramName}] не существует.";
15          private static string FormatMessage(TLinkAddress link) => $"Связь [{link}] переданная в
16             ↪ качестве аргумента не существует.";
17      }
18 }

```

## ./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          public ArgumentLinkHasDependenciesException(TLinkAddress link) :
12             ↪ base(FormatMessage(link)) { }
13          private static string FormatMessage(TLinkAddress link, string paramName) => $"У связи
14             ↪ [{link}] переданной в аргумент [{paramName}] присутствуют зависимости, которые
15             ↪ препятствуют изменению её внутренней структуры.";
16          private static string FormatMessage(TLinkAddress link) => $"У связи [{link}] переданной
17             ↪ в качестве аргумента присутствуют зависимости, которые препятствуют изменению её
18             ↪ внутренней структуры.";
19      }
20 }

```

## ./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> : Exception
8      {
9          public static readonly string DefaultMessage = "Достигнут лимит количества связей в
10             ↪ хранилище.";
11          public LinksLimitReachedException(string message) : base(message) { }
12          public LinksLimitReachedException(TLinkAddress limit) : this(FormatMessage(limit)) { }
13          public LinksLimitReachedException() : base(DefaultMessage) { }
14          private static string FormatMessage(TLinkAddress limit) => $"Достигнут лимит количества
15             ↪ связей в хранилище ({limit}).";
16      }
17 }

```

## ./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 : Exception
8      {
9          public static readonly string DefaultMessage = "Связь с таким же значением уже
10             ↪ существует.";
11          public LinkWithSameValueAlreadyExistsException(string message) : base(message) { }
12          public LinkWithSameValueAlreadyExistsException() : base(DefaultMessage) { }
13      }
14 }

```

./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(IList<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>
46        /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
47        /// → может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
48        /// → Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
49        /// <returns>True, в случае если проход по связям не был прерван и False в обратном
50        /// → случае.</returns>
51        TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler, IList<TLinkAddress>
52        → restrictions);
53
54        #endregion
55
56        #region Write
57
58        /// <summary>
59        /// Создаёт связь.
60        /// </summary>
61        /// <returns>Индекс созданной связи.</returns>
62        TLinkAddress Create(IList<TLinkAddress> restrictions); // TODO: Возможно всегда нужно
63        → принимать restrictions, возможно и возвращать связь нужно целиком.
64
65        /// <summary>
66        /// Обновляет связь с указанными restrictions[Constants.IndexPart] в адресом связи
67        /// на связь с указанным новым содержимым.
68        /// </summary>
69        /// <param name="restrictions">
70        /// Ограничения на содержимое связей.
71        /// Предполагается, что будет указан индекс связи (в restrictions[Constants.IndexPart])
72        /// → и далее за ним будет следовать содержимое связи.
73        /// Каждое ограничение может иметь значения: Constants.Null - 0-я связь, обозначающая
74        /// → ссылку на пустоту,
75        /// Constants.Itself - требование установить ссылку на себя, 1..∞ конкретный индекс
76        /// → другой связи.
77        /// </param>
78        /// <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 }

```

# ./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             => Comparer<TLinkAddress>.Default.Compare(links.Count(link), default) > 0;
28
29         /// <param name="links">Хранилище связей.</param>
30         /// <param name="link">Индекс проверяемой на существование связи.</param>
31         /// <remarks>
32         /// TODO: May be move to EnsureExtensions or make it both there and here
33         /// </remarks>
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    → TConstants> links, TLinkAddress link)
36             where TConstants : LinksConstants<TLinkAddress>
37         {
38             if (!links.Exists(link))
39             {
40                 throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link);
41             }
42         }
43
44         /// <param name="links">Хранилище связей.</param>
45         /// <param name="link">Индекс проверяемой на существование связи.</param>
46         /// <param name="argumentName">Имя аргумента, в который передаётся индекс связи.</param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public static void EnsureLinkExists<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    → TConstants> links, TLinkAddress link, string argumentName)
49             where TConstants : LinksConstants<TLinkAddress>
50         {
51             if (!links.Exists(link))
52             {
53                 throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
54             }
55         }
56
57         /// <summary>
58         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    → (handler) для каждой подходящей связи.
59         /// </summary>

```

```

60 /// <param name="links">Хранилище связей.</param>
61 /// <param name="handler">Обработчик каждой подходящей связи.</param>
62 /// <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>
63 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public static TLinkAddress Each<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, Func<IList<TLinkAddress>, TLinkAddress> handler, params
    ↳ TLinkAddress[] restrictions)
66     where TConstants : LinksConstants<TLinkAddress>
67     => links.Each(handler, restrictions);
68
69 /// <summary>
70 /// Возвращает части-значения для связи с указанным индексом.
71 /// </summary>
72 /// <param name="links">Хранилище связей.</param>
73 /// <param name="link">Индекс связи.</param>
74 /// <returns>Уникальную связь.</returns>
75 [MethodImpl(MethodImplOptions.AggressiveInlining)]
76 public static IList<TLinkAddress> GetLink<TLinkAddress, TConstants>(this
    ↳ ILinks<TLinkAddress, TConstants> links, TLinkAddress link)
77     where TConstants : LinksConstants<TLinkAddress>
78 {
79     var constants = links.Constants;
80     var linkPartsSetter = new Setter<IList<TLinkAddress>,
    ↳ TLinkAddress>(constants.Continue, constants.Break);
81     links.Each(linkPartsSetter.SetAndReturnTrue, link);
82     return linkPartsSetter.Result;
83 }
84
85 #region Points
86
87 /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
    ↳ точкой полностью (связью замкнутой на себе дважды).</summary>
88 /// <param name="links">Хранилище связей.</param>
89 /// <param name="link">Индекс проверяемой связи.</param>
90 /// <returns>Значение, определяющее является ли связь точкой полностью.</returns>
91 /// <remarks>
92 /// Связь точка - это связь, у которой начало (Source) и конец (Target) есть сама эта
    ↳ связь.
93 /// Но что, если точка уже есть, а нужно создать пару с таким же значением? Должны ли
    ↳ точка и пара существовать одновременно?
94 /// Или в качестве решения для точек нужно использовать 0 в качестве начала и конца, а
    ↳ сортировать по индексу в массиве связей?
95 /// Какое тогда будет значение Source и Target у точки? 0 или её индекс?
96 /// Или точка должна быть одновременно точкой и парой, а также последовательностями из
    ↳ самой себя любого размера?
97 /// Как только есть ссылка на себя, появляется этот парадокс, причём достаточно даже
    ↳ одной ссылки на себя (частичной точки).
98 /// А что если не выбирать что является точкой, пара нулей (цикл через пустоту) или
99 /// самостоятельный цикл через себя? Что если предоставить все варианты использования
    ↳ связей?
100 /// Что если разрешить и нули, а так же частичные варианты?
101 ///
102 /// Что если точка, это только в том случае когда link.Source == link &&
    ↳ link.Target == link , т.е. дважды ссылка на себя.
103 /// А пара это тогда, когда link.Source == link.Target && link.Source != link ,
    ↳ т.е. ссылка не на себя а во вне.
104 ///
105 /// Тогда если у нас уже создана пара, но нам нужна точка, мы можем используя
    ↳ промежуточную связь,
106 /// например "DoubletOf" обозначить что является точно парой, а что точно точкой.
107 /// И наоборот этот же метод поможет, если уже существует точка, но нам нужна пара.
108 /// </remarks>
109 public static bool IsFullPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
    ↳ TConstants> links, TLinkAddress link)
110     where TConstants : LinksConstants<TLinkAddress>
111 {
112     links.EnsureLinkExists(link);
113     return Point<TLinkAddress>.IsFullPoint(links.GetLink(link));
114 }
115
116 /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
    ↳ точкой частично (связью замкнутой на себе как минимум один раз).</summary>
117 /// <param name="links">Хранилище связей.</param>
118 /// <param name="link">Индекс проверяемой связи.</param>

```

```

119     /// <returns>Значение, определяющее является ли связь точкой частично.</returns>
120     /// <remarks>
121     /// Достаточно любой одной ссылки на себя.
122     /// Также в будущем можно будет проверять и всех родителей, чтобы проверить есть ли
123     /// ↪ ссылки на себя (на эту связь).
124     /// </remarks>
125     public static bool IsPartialPoint<TLinkAddress, TConstants>(this ILinks<TLinkAddress,
126     ↪ TConstants> links, TLinkAddress link)
127     where TConstants : LinksConstants<TLinkAddress>
128     {
129         links.EnsureLinkExists(link);
130         return Point<TLinkAddress>.IsPartialPoint(links.GetLink(link));
131     }
132 }
133 }

./ISynchronizedLinks.cs
1 using Platform.Threading.Synchronization;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data
6 {
7     public interface ISynchronizedLinks<TLinkAddress, TLinks, TConstants> :
8     ↪ ISynchronized<TLinks>, ILinks<TLinkAddress, TConstants>
9     where TLinks : ILinks<TLinkAddress, TConstants>
10     where TConstants : LinksConstants<TLinkAddress>
11     {
12     }
13 }

./LinkAddress.cs
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data
8 {
9     public struct LinkAddress<TLinkAddress> : IEquatable<LinkAddress<TLinkAddress>>,
10     ↪ IList<TLinkAddress>
11     {
12         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
13         ↪ EqualityComparer<TLinkAddress>.Default;
14
15         public readonly TLinkAddress Index;
16
17         public LinkAddress(TLinkAddress index) => Index = index;
18
19         public TLinkAddress this[int index]
20         {
21             get
22             {
23                 if (index == 0)
24                 {
25                     return Index;
26                 }
27                 else
28                 {
29                     throw new IndexOutOfRangeException();
30                 }
31             }
32             set => throw new NotSupportedException();
33         }
34
35         public int Count => 1;
36
37         public bool IsReadOnly => true;
38
39         public void Add(TLinkAddress item) => throw new NotSupportedException();
40
41         public void Clear() => throw new NotSupportedException();
42
43         public bool Contains(TLinkAddress item) => _equalityComparer.Equals(item, Index) ? true
44         ↪ : false;
45
46         public void CopyTo(TLinkAddress[] array, int arrayIndex) => array[arrayIndex] = Index;

```

```

44
45     public IEnumerator<TLinkAddress> GetEnumerator()
46     {
47         yield return Index;
48     }
49
50     public int IndexOf(TLinkAddress item) => _equalityComparer.Equals(item, Index) ? 0 : -1;
51
52     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
53
54     public bool Remove(TLinkAddress item) => throw new NotSupportedException();
55
56     public void RemoveAt(int index) => throw new NotSupportedException();
57
58     IEnumerator IEnumerable.GetEnumerator()
59     {
60         yield return Index;
61     }
62
63     public bool Equals(LinkAddress<TLinkAddress> other) => _equalityComparer.Equals(Index,
64     ↪ other.Index);
65
66     public static implicit operator TLinkAddress(LinkAddress<TLinkAddress> linkAddress) =>
67     ↪ linkAddress.Index;
68
69     public static implicit operator LinkAddress<TLinkAddress>(TLinkAddress linkAddress) =>
70     ↪ new LinkAddress<TLinkAddress>(linkAddress);
71
72     public override bool Equals(object obj) => obj is LinkAddress<TLinkAddress> linkAddress
73     ↪ ? Equals(linkAddress) : false;
74
75     public override int GetHashCode() => Index.GetHashCode();
76
77     public override string ToString() => Index.ToString();
78 }
79
80 }

```

#### ./LinksConstants.cs

```

1  using Platform.Numbers;
2  using Platform.Ranges;
3  using Platform.Reflection;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data
8  {
9      public class LinksConstants<TLinkAddress>
10     {
11         public static readonly int DefaultTargetPart = 2;
12
13         #region Link parts
14
15         /// <summary>Возвращает индекс части, которая отвечает за индекс (адрес, идентификатор)
16         ↪ самой связи.</summary>
17         public int IndexPart { get; }
18
19         /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-начало (первая
20         ↪ часть-значение).</summary>
21         public int SourcePart { get; }
22
23         /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-конец
24         ↪ (последняя часть-значение).</summary>
25         public int TargetPart { get; }
26
27         #endregion
28
29         #region Flow control
30
31         /// <summary>Возвращает значение, обозначающее продолжение прохода по связям.</summary>
32         /// <remarks>Используется в функции обработчике, который передаётся в функцию
33         ↪ 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
41         ↪ Each.</remarks>
42         public TLinkAddress Break { get; }
43     }
44 }

```

```

38
39 #endregion
40
41 #region Special symbols
42
43 /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
44 public TLinkAddress Null { get; }
45
46 /// <summary>Возвращает значение, обозначающее любую связь.</summary>
47 /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
48   ↳ создавать все варианты последовательностей в функции Create.</remarks>
49 public TLinkAddress Any { get; }
50
51 /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
52 public TLinkAddress Itself { get; }
53
54 #endregion
55
56 #region References
57
58 /// <summary>Возвращает диапазон возможных индексов для внутренних связей (внутренних
59   ↳ ссылок).</summary>
60 public Range<TLinkAddress> PossibleInnerReferencesRange { get; }
61
62 /// <summary>Возвращает диапазон возможных индексов для внешних связей (внешних
63   ↳ ссылок).</summary>
64 public Range<TLinkAddress>? PossibleExternalReferencesRange { get; }
65
66 #endregion
67
68 public LinksConstants(int targetPart, Range<TLinkAddress> possibleInnerReferencesRange,
69   ↳ Range<TLinkAddress>? possibleExternalReferencesRange)
70 {
71     IndexPart = 0;
72     SourcePart = 1;
73     TargetPart = targetPart;
74     Null = Integer<TLinkAddress>.Zero;
75     Break = Integer<TLinkAddress>.Zero;
76     var currentInnerReferenceIndex = possibleInnerReferencesRange.Maximum;
77     Continue = currentInnerReferenceIndex;
78     Decrement(ref currentInnerReferenceIndex);
79     Skip = currentInnerReferenceIndex;
80     Decrement(ref currentInnerReferenceIndex);
81     Any = currentInnerReferenceIndex;
82     Decrement(ref currentInnerReferenceIndex);
83     Itself = currentInnerReferenceIndex;
84     Decrement(ref currentInnerReferenceIndex);
85     PossibleInnerReferencesRange = (possibleInnerReferencesRange.Minimum,
86   ↳ currentInnerReferenceIndex);
87     PossibleExternalReferencesRange = possibleExternalReferencesRange;
88 }
89
90 private static void Decrement(ref TLinkAddress currentInnerReferenceIndex) =>
91   ↳ currentInnerReferenceIndex = Arithmetic.Decrement(currentInnerReferenceIndex);
92
93 public LinksConstants(Range<TLinkAddress> possibleInnerReferencesRange,
94   ↳ Range<TLinkAddress>? possibleExternalReferencesRange) : this(DefaultTargetPart,
95   ↳ possibleInnerReferencesRange, possibleExternalReferencesRange) { }
96
97 public LinksConstants(int targetPart, Range<TLinkAddress> possibleInnerReferencesRange)
98   ↳ : this(targetPart, possibleInnerReferencesRange, null) { }
99
100 public LinksConstants(Range<TLinkAddress> possibleInnerReferencesRange) :
101   ↳ this(DefaultTargetPart, possibleInnerReferencesRange, null) { }
102
103 public LinksConstants() : this(DefaultTargetPart, (Integer<TLinkAddress>.One,
104   ↳ NumericType<TLinkAddress>.MaxValue), null) { }
105
106 }
107
108 }

```

./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>
    ↪ linksConstants, TLinkAddress address) => linksConstants.IsInnerReference(address) ||
    ↪ linksConstants.IsExternalReference(address);
11
12     [MethodImpl(MethodImplOptions.AggressiveInlining)]
13     public static bool IsInnerReference<TLinkAddress>(this LinksConstants<TLinkAddress>
    ↪ linksConstants, TLinkAddress address) =>
    ↪ linksConstants.PossibleInnerReferencesRange.ContainsValue(address);
14
15     [MethodImpl(MethodImplOptions.AggressiveInlining)]
16     public static bool IsExternalReference<TLinkAddress>(this LinksConstants<TLinkAddress>
    ↪ linksConstants, TLinkAddress address) =>
    ↪ linksConstants.PossibleExternalReferencesRange?.ContainsValue(address) ?? false;
17 }
18 }

```

## ./Point.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Exceptions;
4  using Platform.Ranges;
5  using Platform.Collections;
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 Point<TLinkAddress>
12     {
13         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
    ↪ EqualityComparer<TLinkAddress>.Default;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static bool IsFullPoint(params TLinkAddress[] link) =>
    ↪ IsFullPoint((IList<TLinkAddress>)link);
17
18         public static bool IsFullPoint(IList<TLinkAddress> link)
19         {
20             Ensure.Always.ArgumentNotEmpty(link, nameof(link));
21             Ensure.Always.ArgumentInRange(link.Count, new Range<int>(2, int.MaxValue),
    ↪ nameof(link), "Cannot determine link's pointness using only its identifier.");
22             return IsFullPointUnchecked(link);
23         }
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         public static bool IsFullPointUnchecked(IList<TLinkAddress> link)
27         {
28             var result = true;
29             for (var i = 1; result && i < link.Count; i++)
30             {
31                 result = _equalityComparer.Equals(link[0], link[i]);
32             }
33             return result;
34         }
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static bool IsPartialPoint(params TLinkAddress[] link) =>
    ↪ IsPartialPoint((IList<TLinkAddress>)link);
38
39         public static bool IsPartialPoint(IList<TLinkAddress> link)
40         {
41             Ensure.Always.ArgumentNotEmpty(link, nameof(link));
42             Ensure.Always.ArgumentInRange(link.Count, new Range<int>(2, int.MaxValue),
    ↪ nameof(link), "Cannot determine link's pointness using only its identifier.");
43             return IsPartialPointUnchecked(link);
44         }
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static bool IsPartialPointUnchecked(IList<TLinkAddress> link)
48         {
49             var result = false;
50             for (var i = 1; !result && i < link.Count; i++)
51             {
52                 result = _equalityComparer.Equals(link[0], link[i]);
53             }
54             return result;
55         }
56     }
57 }

```



## ./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  }
```

## ./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 }
```

## ./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     /// При оптимизации можно использовать встроенную поддержку стеков в процессор.
15     ///
16     /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
17     /// Решить встраивать ли защиту от заикливания.
18     /// Альтернативой защиты от закливания может быть заранее известное ограничение на
19     /// ↪ погружение вглубь.
20     /// А так же качественное распознавание прохода по циклическому графу.
21     /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
22     /// ↪ стека.
23     /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
24     /// </remarks>
25     public static class SequenceWalker
26     {
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static void WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
29             ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
30             ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
31         {
32             var stack = new Stack<TLinkAddress>();
33             var element = sequence;
34             if (isElement(element))
35             {
36                 visit(element);
37             }
38             else
39             {
40                 while (true)
41                 {
42                     if (isElement(element))
43                     {
44                         if (stack.Count == 0)
45                         {
46                             break;
47                         }
48                         element = stack.Pop();
49                         var source = getSource(element);
50                         var target = getTarget(element);
51                         if (isElement(source))
52                         {
53                             visit(source);
54                         }
55                     }
56                     else
57                     {
58                         stack.Push(element);
59                         element = target;
60                     }
61                 }
62             }
63         }
64     }
65 }
```

```

50     }
51     if (isElement(target))
52     {
53         visit(target);
54     }
55     element = target;
56 }
57 else
58 {
59     stack.Push(element);
60     element = getSource(element);
61 }
62 }
63 }
64 }
65
66 [MethodImpl(MethodImplOptions.AggressiveInlining)]
67 public static void WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
68 {
69     var stack = new Stack<TLinkAddress>();
70     var element = sequence;
71     if (isElement(element))
72     {
73         visit(element);
74     }
75     else
76     {
77         while (true)
78         {
79             if (isElement(element))
80             {
81                 if (stack.Count == 0)
82                 {
83                     break;
84                 }
85                 element = stack.Pop();
86                 var source = getSource(element);
87                 var target = getTarget(element);
88                 if (isElement(target))
89                 {
90                     visit(target);
91                 }
92                 if (isElement(source))
93                 {
94                     visit(source);
95                 }
96                 element = source;
97             }
98             else
99             {
100                 stack.Push(element);
101                 element = getTarget(element);
102             }
103         }
104     }
105 }
106 }
107 }

```

## ./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     /// При оптимизации можно использовать встроенную поддержку стеков в процессор.
15     ///
16     /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?

```

```

17  /// Решить встраивать ли защиту от закикливания.
18  /// Альтернативой защиты от закикливания может быть заранее известное ограничение на
    ↳ погружение вглубь.
19  /// А так же качественное распознавание прохода по циклическому графу.
20  /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
    ↳ стека.
21  /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
22  /// </remarks>
23  public static class StopableSequenceWalker
24  {
25      [MethodImpl(MethodImplOptions.AggressiveInlining)]
26      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)
27  {
28      var exited = 0;
29      var stack = new Stack<TLinkAddress>();
30      var element = sequence;
31      if (isElement(element))
32      {
33          return visit(element);
34      }
35      while (true)
36      {
37          if (isElement(element))
38          {
39              if (stack.Count == 0)
40              {
41                  return true;
42              }
43              element = stack.Pop();
44              exit(element);
45              exited++;
46              var source = getSource(element);
47              var target = getTarget(element);
48              if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
    ↳ !visit(source))
49              {
50                  return false;
51              }
52              if ((isElement(target) || !canEnter(target)) && !visit(target))
53              {
54                  return false;
55              }
56              element = target;
57          }
58          else
59          {
60              if (canEnter(element))
61              {
62                  enter(element);
63                  exited = 0;
64                  stack.Push(element);
65                  element = getSource(element);
66              }
67              else
68              {
69                  if (stack.Count == 0)
70                  {
71                      return true;
72                  }
73                  element = stack.Pop();
74                  exit(element);
75                  exited++;
76                  var source = getSource(element);
77                  var target = getTarget(element);
78                  if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
    ↳ !visit(source))
79                  {
80                      return false;
81                  }
82                  if ((isElement(target) || !canEnter(target)) && !visit(target))
83                  {
84                      return false;
85                  }
86                  element = target;
87              }
88          }

```

```

89     }
90 }
91
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
94 {
95     var stack = new Stack<TLinkAddress>();
96     var element = sequence;
97     if (isElement(element))
98     {
99         return visit(element);
100     }
101     while (true)
102     {
103         if (isElement(element))
104         {
105             if (stack.Count == 0)
106             {
107                 return true;
108             }
109             element = stack.Pop();
110             var source = getSource(element);
111             var target = getTarget(element);
112             if (isElement(source) && !visit(source))
113             {
114                 return false;
115             }
116             if (isElement(target) && !visit(target))
117             {
118                 return false;
119             }
120             element = target;
121         }
122         else
123         {
124             stack.Push(element);
125             element = getSource(element);
126         }
127     }
128 }
129
130 [MethodImpl(MethodImplOptions.AggressiveInlining)]
131 public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
132 {
133     var stack = new Stack<TLinkAddress>();
134     var element = sequence;
135     if (isElement(element))
136     {
137         return visit(element);
138     }
139     while (true)
140     {
141         if (isElement(element))
142         {
143             if (stack.Count == 0)
144             {
145                 return true;
146             }
147             element = stack.Pop();
148             var source = getSource(element);
149             var target = getTarget(element);
150             if (isElement(target) && !visit(target))
151             {
152                 return false;
153             }
154             if (isElement(source) && !visit(source))
155             {
156                 return false;
157             }
158             element = source;
159         }
160         else
161         {
162             stack.Push(element);
163             element = getTarget(element);

```

```

164     }
165 }
166 }
167 }
168 }

```

./Universal/IUniLinksCRUD.cs

```

1 using System;
2
3 // ReSharper disable TypeParameterCanBeVariant
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Universal
7 {
8     /// <remarks>
9     /// CRUD aliases for IUniLinks.
10    /// </remarks>
11    public interface IUniLinksCRUD<TLinkAddress>
12    {
13        TLinkAddress Read(ulong partType, TLinkAddress link);
14        bool Read(Func<TLinkAddress, bool> handler, params TLinkAddress[] pattern);
15        TLinkAddress Create(TLinkAddress[] parts);
16        TLinkAddress Update(TLinkAddress[] before, TLinkAddress[] after);
17        void Delete(TLinkAddress[] parts);
18    }
19 }

```

./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<IList<TLinkAddress>>> Trigger(IList<TLinkAddress> condition,
13            ↳ IList<TLinkAddress> substitution);
14    }
15
16    /// <remarks>Minimal sufficient universal Links API (for step by step operations).</remarks>
17    public partial interface IUniLinks<TLinkAddress>
18    {
19        /// <returns>
20        /// TLinkAddress that represents True (was finished fully) or TLinkAddress that
21        /// ↳ represents False (was stopped).
22        /// This is done to assure ability to push up stop signal through recursion stack.
23        /// </returns>
24        /// <remarks>
25        /// { 0, 0, 0 } => { itself, itself, itself } // create
26        /// { 1, any, any } => { itself, any, 3 } // update
27        /// { 3, any, any } => { 0, 0, 0 } // delete
28        /// </remarks>
29        TLinkAddress Trigger(IList<TLinkAddress> patternOrCondition, Func<IList<TLinkAddress>,
30            ↳ TLinkAddress> matchHandler,
31            IList<TLinkAddress> substitution, Func<IList<TLinkAddress>,
32            ↳ IList<TLinkAddress>, TLinkAddress> substitutionHandler);
33
34        TLinkAddress Trigger(IList<TLinkAddress> restriction, Func<IList<TLinkAddress>,
35            ↳ IList<TLinkAddress>, TLinkAddress> matchedHandler,
36            IList<TLinkAddress> substitution, Func<IList<TLinkAddress>, IList<TLinkAddress>,
37            ↳ TLinkAddress> substitutedHandler);
38    }
39
40    /// <remarks>Extended with small optimization.</remarks>
41    public partial interface IUniLinks<TLinkAddress>
42    {
43        /// <remarks>
44        /// Something simple should be simple and optimized.
45        /// </remarks>
46        TLinkAddress Count(IList<TLinkAddress> restrictions);
47    }
48 }

```

./Universal/IUniLinksGS.cs

```
1 using System;
2
3 // ReSharper disable TypeParameterCanBeVariant
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Universal
7 {
8     /// <remarks>
9     /// Get/Set aliases for IUniLinks.
10    /// </remarks>
11    public interface IUniLinksGS<TLinkAddress>
12    {
13        TLinkAddress Get(ulong partType, TLinkAddress link);
14        bool Get(Func<TLinkAddress, bool> handler, params TLinkAddress[] pattern);
15        TLinkAddress Set(TLinkAddress[] before, TLinkAddress[] after);
16    }
17 }
```

./Universal/IUniLinksIO.cs

```
1 using System;
2
3 // ReSharper disable TypeParameterCanBeVariant
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Universal
7 {
8     /// <remarks>
9     /// In/Out aliases for IUniLinks.
10    /// TLinkAddress can be any number type of any size.
11    /// </remarks>
12    public interface IUniLinksIO<TLinkAddress>
13    {
14        /// <remarks>
15        /// default(TLinkAddress) means any link.
16        /// Single element pattern means just element (link).
17        /// Handler gets array of link contents.
18        /// * link[0] is index or identifier.
19        /// * link[1] is source or first.
20        /// * link[2] is target or second.
21        /// * link[3] is linker or third.
22        /// * link[n] is nth part/parent/element/value
23        /// of link (if variable length links used).
24        ///
25        /// Stops and returns false if handler return false.
26        ///
27        /// Acts as Each, Foreach, Select, Search, Match & ...
28        ///
29        /// Handles all links in store if pattern/restrictions is not defined.
30        /// </remarks>
31        bool Out(Func<TLinkAddress[], bool> handler, params TLinkAddress[] pattern);
32
33        /// <remarks>
34        /// default(TLinkAddress) means itself.
35        /// Equivalent to:
36        /// * creation if before == null
37        /// * deletion if after == null
38        /// * update if before != null & & after != null
39        /// * default(TLinkAddress) if before == null & & after == null
40        ///
41        /// Possible interpretation
42        /// * In(null, new[] { }) creates point (link that points to itself using minimum number
43        ///   ↳ of parts).
44        /// * In(new[] { 4 }, null) deletes 4th link.
45        /// * In(new[] { 4 }, new [] { 5 }) delete 5th link if it exists and moves 4th link to
46        ///   ↳ 5th index.
47        /// * In(new[] { 4 }, new [] { 0, 2, 3 }) replaces 4th link with new doublet link (with
48        ///   ↳ 2 as source and 3 as target), 0 means it can be placed in any address.
49        /// ...
50        /// </remarks>
51        TLinkAddress In(TLinkAddress[] before, TLinkAddress[] after);
52    }
53 }
```

./Universal/IUniLinksIOWithExtensions.cs

```
1 // ReSharper disable TypeParameterCanBeVariant
2 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
3
```

```

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

```

./Universal/IUniLinksRW.cs

```

1 using System;
2
3 // ReSharper disable TypeParameterCanBeVariant
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Universal
7 {
8     /// <remarks>
9     /// Read/Write aliases for IUniLinks.
10    /// </remarks>
11    public interface IUniLinksRW<TLinkAddress>
12    {
13        TLinkAddress Read(ulong partType, TLinkAddress link);
14        bool Read(Func<TLinkAddress, bool> handler, params TLinkAddress[] pattern);
15        TLinkAddress Write(TLinkAddress[] before, TLinkAddress[] after);
16    }
17 }

```

## Index

- ./Exceptions/ArgumentLinkDoesNotExistsException.cs, 1
- ./Exceptions/ArgumentLinkHasDependenciesException.cs, 1
- ./Exceptions/LinkWithSameValueAlreadyExistsException.cs, 1
- ./Exceptions/LinksLimitReachedException.cs, 1
- ./ILinks.cs, 1
- ./ILinksExtensions.cs, 3
- ./ISynchronizedLinks.cs, 5
- ./LinkAddress.cs, 5
- ./LinksConstants.cs, 6
- ./LinksConstantsExtensions.cs, 7
- ./Point.cs, 8
- ./Sequences/ISequenceAppender.cs, 9
- ./Sequences/ISequenceWalker.cs, 9
- ./Sequences/SequenceWalker.cs, 9
- ./Sequences/StopableSequenceWalker.cs, 10
- ./Universal/IUniLinks.cs, 13
- ./Universal/IUniLinksCRUD.cs, 13
- ./Universal/IUniLinksGS.cs, 13
- ./Universal/IUniLinksIO.cs, 14
- ./Universal/IUniLinksIOWithExtensions.cs, 14
- ./Universal/IUniLinksRW.cs, 15