

LinksPlatform's Platform.Data Class Library

./Constants/ILinksAddressConstants.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Constants
4  {
5      public interface ILinksAddressConstants<out TAddress>
6      {
7          /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
8          TAddress Null { get; }
9
10         /// <summary>Возвращает значение, обозначающее любую связь.</summary>
11         /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
12         ↪ создавать все варианты последовательностей в функции Create.</remarks>
13         TAddress Any { get; }
14
15         /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
16         TAddress Itself { get; }
17
18         /// <summary>Возвращает минимально возможный индекс существующей связи.</summary>
19         TAddress MinPossibleIndex { get; }
20
21         /// <summary>Возвращает максимально возможный индекс существующей связи.</summary>
22         /// <remarks>
23         /// Если за каждую константу будет отвечать отдельная связь, диапазон возможных связей
24         ↪ будет уменьшен.
25         /// Благодаря логике конвертации Integer для каждого типа здесь будет максимальное
26         ↪ значение этого типа.
27         /// </remarks>
28         TAddress MaxPossibleIndex { get; }
29     }
30 }
```

./Constants/ILinksCombinedConstants.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Constants
4  {
5      public interface ILinksCombinedConstants<TDecision, TAddress, TPartIndex, TLinksConstants> :
6          ILinksDecisionConstants<TDecision>,
7          ILinksAddressConstants<TAddress>,
8          ILinksPartConstants<TPartIndex>
9      where TLinksConstants : ILinksDecisionConstants<TDecision>,
10         ↪ ILinksAddressConstants<TAddress>, ILinksPartConstants<TPartIndex>
11     {
12     }
13 }
```

./Constants/ILinksDecisionConstants.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Constants
4  {
5      public interface ILinksDecisionConstants<out TDecision>
6      {
7          /// <summary>Возвращает булевское значение, обозначающее продолжение прохода по
8          ↪ связям.</summary>
9          /// <remarks>Используется в функции обработчике, который передаётся в функцию
10          ↪ Each.</remarks>
11          TDecision Continue { get; }
12
13          /// <summary>Возвращает булевское значение, обозначающее остановку прохода по
14          ↪ связям.</summary>
15          /// <remarks>Используется в функции обработчике, который передаётся в функцию
16          ↪ Each.</remarks>
17          TDecision Break { get; }
18     }
19 }
```

./Constants/ILinksPartConstants.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Constants
4  {
5      public interface ILinksPartConstants<out TPartIndex>
6      {
7          /// <summary>Возвращает индекс части, которая отвечает за индекс (адрес, идентификатор)
8          ↪ самой связи.</summary>
9      }
10 }
```

```

8         TPartIndex IndexPart { get; }
9
10        /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-начало (первая
    ↳ часть-значение).</summary>
11        TPartIndex SourcePart { get; }
12
13        /// <summary>Возвращает индекс части, которая отвечает за ссылку на связь-конец
    ↳ (последняя часть-значение).</summary>
14        TPartIndex TargetPart { get; }
15    }
16 }

```

./Constants/LinksCombinedConstants[TDecision, TAddress].cs

```

1  using Platform.Numbers;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Constants
6  {
7      public class LinksCombinedConstants<TDecision, TAddress> :
    ↳ LinksDecisionConstants<TDecision>, ILinksAddressConstants<TAddress>
8      {
9          /// <summary>Возвращает значение, обозначающее отсутствие связи.</summary>
10         public TAddress Null { get; }
11
12         /// <summary>Возвращает значение, обозначающее любую связь.</summary>
13         /// <remarks>Возможно нужно зарезервировать отдельное значение, тогда можно будет
    ↳ создавать все варианты последовательностей в функции Create.</remarks>
14         public TAddress Any { get; }
15
16         /// <summary>Возвращает значение, обозначающее связь-ссылку на саму связь.</summary>
17         public TAddress Itself { get; }
18
19         /// <summary>Возвращает минимально возможный индекс существующей связи.</summary>
20         public TAddress MinPossibleIndex { get; }
21
22         /// <summary>Возвращает максимально возможный индекс существующей связи.</summary>
23         /// <remarks>
24         /// Если за каждую константу будет отвечать отдельная связь, диапазон возможных связей
    ↳ будет уменьшен.
25         /// Благодаря логике конвертации Integer для каждого типа здесь будет максимальное
    ↳ значение этого типа.
26         /// </remarks>
27         public TAddress MaxPossibleIndex { get; }
28
29         public LinksCombinedConstants()
30         {
31             Null = Integer<TAddress>.Zero;
32             MinPossibleIndex = Integer<TAddress>.One;
33             MaxPossibleIndex = Arithmetic.Subtract<TAddress>(ulong.MaxValue, 3);
34             Itself = Arithmetic.Subtract<TAddress>(ulong.MaxValue, 2);
35             Any = Arithmetic.Subtract<TAddress>(ulong.MaxValue, 1);
36             // ulong.MaxValue is reserved for "Continue"
37         }
38     }
39 }

```

./Constants/LinksCombinedConstants[TDecision, TAddress, TPartIndex].cs

```

1  using Platform.Numbers;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Constants
6  {
7      public class LinksCombinedConstants<TDecision, TAddress, TPartIndex> :
    ↳ LinksCombinedConstants<TDecision, TAddress>, ILinksCombinedConstants<TDecision,
    ↳ TAddress, TPartIndex, LinksCombinedConstants<TDecision, TAddress, TPartIndex>>
8      {
9          public TPartIndex IndexPart { get; } = Integer<TPartIndex>.Zero;
10         public TPartIndex SourcePart { get; } = Integer<TPartIndex>.One;
11         public TPartIndex TargetPart { get; } = Integer<TPartIndex>.Two;
12     }
13 }

```

./Constants/LinksDecisionConstants.cs

```

1  using Platform.Numbers;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4

```

```

5 namespace Platform.Data.Constants
6 {
7     /// <remarks>
8     /// Возможно каждая константа должна иметь своё уникальное значение (которое можно
9     /// ↳ отсчитывать от конца доступных значений),
10    /// например (ulong.MaxValue - 1) и т.п.
11    /// </remarks>
12    public class LinksDecisionConstants<TDecision> : ILinksDecisionConstants<TDecision>
13    {
14        // Cannot be supported anymore because of MathHelpers.Subtract usage. Cannot operate IL
15        // ↳ subtract directly on Integer. If needed later special code should be emitter for
16        // ↳ this case.
17        //public readonly LinksConstants<Integer, Integer, Integer> Auto =
18        // ↳ Default<LinksConstants<Integer, Integer, Integer>>.Instance;
19
20        /// <summary>Возвращает булевское значение, обозначающее продолжение прохода по
21        /// ↳ связям.</summary>
22        /// <remarks>Используется в функции обработчике, который передаётся в функцию
23        /// ↳ Each.</remarks>
24        public TDecision Continue { get; } = (Integer<TDecision>)ulong.MaxValue;
25
26        /// <summary>Возвращает булевское значение, обозначающее остановку прохода по
27        /// ↳ связям.</summary>
28        /// <remarks>Используется в функции обработчике, который передаётся в функцию
29        /// ↳ Each.</remarks>
30        public TDecision Break { get; }
31    }
32 }

```

./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<TLink> : ArgumentException
8     {
9         public ArgumentLinkDoesNotExistsException(TLink link, string paramName) :
10             ↳ base(FormatMessage(link, paramName), paramName) { }
11         public ArgumentLinkDoesNotExistsException(TLink link) : base(FormatMessage(link)) { }
12         private static string FormatMessage(TLink link, string paramName) => $"Связь [{link}]
13             ↳ переданная в аргумент [{paramName}] не существует.";
14         private static string FormatMessage(TLink link) => $"Связь [{link}] переданная в
15             ↳ качестве аргумента не существует.";
16     }
17 }

```

./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<TLink> : ArgumentException
8     {
9         public ArgumentLinkHasDependenciesException(TLink link, string paramName) :
10             ↳ base(FormatMessage(link, paramName), paramName) { }
11         public ArgumentLinkHasDependenciesException(TLink link) : base(FormatMessage(link)) { }
12         private static string FormatMessage(TLink link, string paramName) => $"У связи [{link}]
13             ↳ переданной в аргумент [{paramName}] присутствуют зависимости, которые препятствуют
14             ↳ изменению её внутренней структуры.";
15         private static string FormatMessage(TLink link) => $"У связи [{link}] переданной в
16             ↳ качестве аргумента присутствуют зависимости, которые препятствуют изменению её
17             ↳ внутренней структуры.";
18     }
19 }

```

./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 : Exception
8     {

```

```

9         public static readonly string DefaultMessage = "Достигнут лимит количества связей в
            ↳ хранилище.";
10        public LinkLimitReachedException(string message) : base(message) { }
11        public LinkLimitReachedException(ulong limit) : this(FormatMessage(limit)) { }
12        public LinkLimitReachedException() : base(DefaultMessage) { }
13        private static string FormatMessage(ulong limit) => $"Достигнут лимит количества связей
            ↳ в хранилище ({limit}).";
14    }
15 }

```

./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
            ↳ существует.";
10        public LinkWithSameValueAlreadyExistsException(string message) : base(message) { }
11        public LinkWithSameValueAlreadyExistsException() : base(DefaultMessage) { }
12    }
13 }

```

./ILinks.cs

```

1 using System;
2 using System.Collections.Generic;
3 using Platform.Data.Constants;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data
8 {
9     /// <summary>
10    /// Представляет интерфейс для работы с данными в формате Links (хранилища взаимосвязей).
11    /// </summary>
12    /// <remarks>
13    /// Этот интерфейс в данный момент не зависит от размера содержимого связи, а значит
14    ↳ подходит как для дуплетов, так и для триплетов и т.п.
15    /// Возможно этот интерфейс подходит даже для Sequences.
16    /// </remarks>
17    public interface ILinks<TLink, TConstants>
18        where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
19    {
20        #region Constants
21
22        /// <summary>
23        /// Возвращает набор констант, который необходим для эффективной коммуникации с методами
24        ↳ этого интерфейса.
25        /// Эти константы не меняются с момента создания точки доступа к хранилищу.
26        /// </summary>
27        TConstants Constants { get; }
28
29        #endregion
30
31        #region Read
32
33        /// <summary>
34        /// Подсчитывает и возвращает общее число связей находящихся в хранилище,
35        ↳ соответствующих указанным ограничениям.
36        /// </summary>
37        /// <param name="restriction">Ограничения на содержимое связей.</param>
38        /// <returns>Общее число связей находящихся в хранилище, соответствующих указанным
39        ↳ ограничениям.</returns>
40        TLink Count(IList<TLink> restriction);
41
42        /// <summary>
43        /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
44        ↳ (handler) для каждой подходящей связи.
45        /// </summary>
46        /// <param name="handler">Обработчик каждой подходящей связи.</param>
47        /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
48        ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
49        ↳ Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
50        /// <returns>True, в случае если проход по связям не был прерван и False в обратном
51        ↳ случае.</returns>
52        TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions);
53    }
54 }

```

```

45 // TODO: Move to UniLinksExtensions
46 //return Trigger(restrictions, (before, after) => handler(before), null, null);
47 // Trigger(restrictions, null, restrictions, null); - Должно быть синонимом
48
49 #endregion
50
51 #region Write
52
53
54 /// <summary>
55 /// Создаёт связь.
56 /// </summary>
57 /// <returns>Индекс созданной связи.</returns>
58 TLink Create(); // TODO: Возможно всегда нужно принимать restrictions, возможно и
    ↳ возвращать связь нужно целиком.
59
60 // TODO: Move to UniLinksExtensions
61 // { 0, 0, 0 } => { ifself, 0, 0 }
62 // { 0 } => { ifself, 0, 0 } *
63
64 //T result = default(T);
65
66 //Func<IList<T>, IList<T>, T> substitutedHandler = (before, after) =>
67 //{
68 //    result = after[Constants.IndexPart];
69 //    return Constants.Continue;
70 //};
71
72 // Сейчас будет полагать что соответствие шаблону (ограничению) произойдёт только один
73 ↳ раз
74 //Trigger(new[] { Constants.Null }, null,
75 //    new[] { Constants.Itself, Constants.Null, Constants.Null },
76 //    substitutedHandler);
77
78 // TODO: Возможна реализация опционального поведения (один ноль-пустота, бесконечность
79 ↳ полей-пустот)
80 // 0 => 1
81
82 // 0 => 1
83 // 0 => 2
84 // 0 => 3
85 // ...
86
87 /// <summary>
88 /// Обновляет связь с указанными restrictions[Constants.IndexPart] в адресом связи
89 /// на связь с указанным новым содержимым.
90 /// </summary>
91 /// <param name="restrictions">
92 /// Ограничения на содержимое связей.
93 /// Предполагается, что будет указан индекс связи (в restrictions[Constants.IndexPart])
94 ↳ и далее за ним будет следовать содержимое связи.
95 /// Каждое ограничение может иметь значения: Constants.Null - 0-я связь, обозначающая
96 ↳ ссылку на пустоту,
97 /// Constants.Itself - требование установить ссылку на себя, 1.. $\infty$  конкретный индекс
98 ↳ другой связи.
99 /// </param>
100 /// <returns>Индекс обновлённой связи.</returns>
101 TLink Update(IList<TLink> restrictions); // TODO: Возможно и возвращать связь нужно
    ↳ целиком.
102
103 // TODO: Move to UniLinksExtensions
104 // { 1, any, any } => { 1, x, y }
105 // { 1 } => { 1, x, y } *
106 // { 1, 3, 4 }
107
108 //Trigger(new[] { restrictions[Constants.IndexPart] }, null,
109 //    new[] { restrictions[Constants.IndexPart], restrictions[Constants.SourcePart],
110 //    ↳ restrictions[Constants.TargetPart] }, null);
111
112 //return restrictions[Constants.IndexPart];
113
114 /// <summary>Удаляет связь с указанным индексом.</summary>
115 /// <param name="link">Индекс удаляемой связи.</param>
116 void Delete(TLink link); // TODO: Возможно всегда нужно принимать restrictions, а так же
    ↳ возвращать удалённую связь, если удаление было реально выполнено, и Null, если нет.
117
118 // TODO: Move to UniLinksExtensions
119 // { 1 } => { 0, 0, 0 }
120 // { 1 } => { 0 } *

```

```

114 //Trigger(new[] { link }, null,
115 //          new[] { Constants.Null }, null);
116
117 // TODO: Если учесть последние TODO, тогда все функции Create, Update, Delete будут
118 //        иметь один и тот же интерфейс - IList<TLink> Method(IList<TLink> restrictions);, что
119 //        может быть удобно для "Create|Update|Delete" транзакционности, !! но нужна ли такая
120 //        транзакционность? Ведь всё что нужно записывать в транзакцию это изменение с чего в
121 //        во что. Создание это index, 0, 0 -> index, X, Y (и начало отслеживания связи).
122 //        Удаление это всегда index, X, Y -> index, 0, 0 (и прекращение отслеживания связи).
123 //        Обновление - аналогично, но состояние отслеживания не меняется.
124
125 // TODO: Хотя пожалуй, выдавать дополнительное значение в виде True/False вряд ли
126 //        допустимо для Delete. Тогда создание это 0,0,0 -> I,S,T и т.п.
127
128 // TODO: Если все методы, Create, Update, Delete будут и принимать и возвращать
129 //        IList<TLink>, то можно всё заменить одним единым Update, у которого для удаления
130 //        нужно указать исходный индекс связи и Constans.Null в качестве его нового значения
131 //        (возможно будет указано 2 ограничения из 3-х)
132
133 #endregion
134 }
135 }

```

./ILinksExtensions.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Setters;
5  using Platform.Data.Constants;
6  using Platform.Data.Exceptions;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data
11 {
12     public static class ILinksExtensions
13     {
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public static TLink Count<TLink, TConstants>(this ILinks<TLink, TConstants> links,
16             → params TLink[] restrictions)
17             where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
18             => links.Count(restrictions);
19
20         /// <summary>
21         /// Возвращает значение, определяющее существует ли связь с указанным индексом в
22         /// хранилище связей.
23         /// </summary>
24         /// <param name="links">Хранилище связей.</param>
25         /// <param name="link">Индекс проверяемой на существование связи.</param>
26         /// <returns>Значение, определяющее существует ли связь.</returns>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static bool Exists<TLink, TConstants>(this ILinks<TLink, TConstants> links, TLink
29             → link)
30             where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
31             => Comparer<TLink>.Default.Compare(links.Count(link), default) > 0;
32
33         /// <param name="links">Хранилище связей.</param>
34         /// <param name="link">Индекс проверяемой на существование связи.</param>
35         /// <remarks>
36         /// TODO: May be move to EnsureExtensions or make it both there and here
37         /// </remarks>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public static void EnsureLinkExists<TLink, TConstants>(this ILinks<TLink, TConstants>
40             → links, TLink link)
41             where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
42         {
43             if (!links.Exists(link))
44             {
45                 throw new ArgumentLinkDoesNotExistsException<TLink>(link);
46             }
47         }
48
49         /// <param name="links">Хранилище связей.</param>
50         /// <param name="link">Индекс проверяемой на существование связи.</param>
51         /// <param name="argumentName">Имя аргумента, в который передаётся индекс связи.</param>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         public static void EnsureLinkExists<TLink, TConstants>(this ILinks<TLink, TConstants>
54             → links, TLink link, string argumentName)
55             where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
56         {
57             if (!links.Exists(link))

```

```

53     {
54         throw new ArgumentException<TLink>(link, argumentName);
55     }
56 }
57
58 /// <summary>
59 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
60 /// → (handler) для каждой подходящей связи.
61 /// </summary>
62 /// <param name="links">Хранилище связей.</param>
63 /// <param name="handler">Обработчик каждой подходящей связи.</param>
64 /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
65 /// → может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
66 /// → Any - отсутствие ограничения, 1..∞ конкретный индекс связи.</param>
67 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
68 /// → случае.</returns>
69 [MethodImpl(MethodImplOptions.AggressiveInlining)]
70 public static TLink Each<TLink, TConstants>(this ILinks<TLink, TConstants> links,
71     Func<IList<TLink>, TLink> handler, params TLink[] restrictions)
72     where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
73     => links.Each(handler, restrictions);
74
75 /// <summary>
76 /// Возвращает части-значения для связи с указанным индексом.
77 /// </summary>
78 /// <param name="links">Хранилище связей.</param>
79 /// <param name="link">Индекс связи.</param>
80 /// <returns>Уникальную связь.</returns>
81 [MethodImpl(MethodImplOptions.AggressiveInlining)]
82 public static IList<TLink> GetLink<TLink, TConstants>(this ILinks<TLink, TConstants>
83     links, TLink link)
84     where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
85 {
86     var constants = links.Constants;
87     var linkPartsSetter = new Setter<IList<TLink>, TLink>(constants.Continue,
88         constants.Break);
89     links.Each(linkPartsSetter.SetAndReturnTrue, link);
90     return linkPartsSetter.Result;
91 }
92
93 #region Points
94
95 /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
96 /// → точкой полностью (связью замкнутой на себе дважды).</summary>
97 /// <param name="links">Хранилище связей.</param>
98 /// <param name="link">Индекс проверяемой связи.</param>
99 /// <returns>Значение, определяющее является ли связь точкой полностью.</returns>
100 /// <remarks>
101 /// Связь точка - это связь, у которой начало (Source) и конец (Target) есть сама эта
102 /// → связь.
103 /// Но что, если точка уже есть, а нужно создать пару с таким же значением? Должны ли
104 /// → точка и пара существовать одновременно?
105 /// Или в качестве решения для точек нужно использовать 0 в качестве начала и конца, а
106 /// → сортировать по индексу в массиве связей?
107 /// Какое тогда будет значение Source и Target у точки? 0 или её индекс?
108 /// Или точка должна быть одновременно точкой и парой, а также последовательностями из
109 /// → самой себя любого размера?
110 /// Как только есть ссылка на себя, появляется этот парадокс, причём достаточно даже
111 /// → одной ссылки на себя (частичной точки).
112 /// А что если не выбирать что является точкой, пара нулей (цикл через пустоту) или
113 /// → самостоятельный цикл через себя? Что если предоставить все варианты использования
114 /// → связей?
115 /// Что если разрешить и нули, а так же частичные варианты?
116 ///
117 /// Что если точка, это только в том случае когда link.Source == link && link.Target == link, т.е. дважды ссылка на себя.
118 /// → link.Target == link, т.е. дважды ссылка на себя.
119 /// А пара это тогда, когда link.Source == link.Target && link.Source != link,
120 /// → т.е. ссылка не на себя а во вне.
121 ///
122 /// Тогда если у нас уже создана пара, но нам нужна точка, мы можем используя
123 /// → промежуточную связь,
124 /// например "DoubletOf" обозначить что является точно парой, а что точно точкой.
125 /// И наоборот этот же метод поможет, если уже существует точка, но нам нужна пара.
126 /// </remarks>
127 public static bool IsFullPoint<TLink, TConstants>(this ILinks<TLink, TConstants> links,
128     TLink link)
129     where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
130 {

```

```

113         links.EnsureLinkExists(link);
114         return Point<TLink>.IsFullPoint(links.GetLink(link));
115     }
116
117     /// <summary>Возвращает значение, определяющее является ли связь с указанным индексом
118     ↪ точкой частично (связью замкнутой на себе как минимум один раз).</summary>
119     /// <param name="links">Хранилище связей.</param>
120     /// <param name="link">Индекс проверяемой связи.</param>
121     /// <returns>Значение, определяющее является ли связь точкой частично.</returns>
122     /// <remarks>
123     /// Достаточно любой одной ссылки на себя.
124     /// Также в будущем можно будет проверять и всех родителей, чтобы проверить есть ли
125     ↪ ссылки на себя (на эту связь).
126     /// </remarks>
127     public static bool IsPartialPoint<TLink, TConstants>(this ILinks<TLink, TConstants>
128     ↪ links, TLink link)
129     where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
130     {
131         links.EnsureLinkExists(link);
132         return Point<TLink>.IsPartialPoint(links.GetLink(link));
133     }
134 }

```

./ISynchronizedLinks.cs

```

1 using Platform.Threading.Synchronization;
2 using Platform.Data.Constants;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data
7 {
8     public interface ISynchronizedLinks<TLink, TLinks, TConstants> : ISynchronized<TLinks>,
9     ↪ ILinks<TLink, TConstants>
10     where TConstants : ILinksCombinedConstants<TLink, TLink, int, TConstants>
11     where TLinks : ILinks<TLink, TConstants>
12     {
13     }
14 }

```

./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<TLink>
12     {
13         private static readonly EqualityComparer<TLink> _equalityComparer =
14         ↪ EqualityComparer<TLink>.Default;
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static bool IsFullPoint(params TLink[] link) => IsFullPoint((IList<TLink>)link);
18
19         public static bool IsFullPoint(IList<TLink> link)
20         {
21             Ensure.Always.ArgumentNotEmpty(link, nameof(link));
22             Ensure.Always.ArgumentInRange(link.Count, new Range<int>(2, int.MaxValue),
23             ↪ nameof(link), "Cannot determine link's pointness using only its identifier.");
24             return IsFullPointUnchecked(link);
25         }
26
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static bool IsFullPointUnchecked(IList<TLink> link)
29         {
30             var result = true;
31             for (var i = 1; result && i < link.Count; i++)
32             {
33                 result = _equalityComparer.Equals(link[0], link[i]);
34             }
35             return result;
36         }
37     }
38 }

```



```

35     [MethodImpl(MethodImplOptions.AggressiveInlining)]
36     public static bool IsPartialPoint(params TLink[] link) =>
37         ↳ IsPartialPoint((IList<TLink>)link);
38
39     public static bool IsPartialPoint(IList<TLink> link)
40     {
41         Ensure.Always.ArgumentNotEmpty(link, nameof(link));
42         Ensure.Always.ArgumentInRange(link.Count, new Range<int>(2, int.MaxValue),
43             ↳ nameof(link), "Cannot determine link's pointness using only its identifier.");
44         return IsPartialPointUnchecked(link);
45     }
46
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public static bool IsPartialPointUnchecked(IList<TLink> link)
49     {
50         var result = false;
51         for (var i = 1; !result && i < link.Count; i++)
52         {
53             result = _equalityComparer.Equals(link[0], link[i]);
54         }
55         return result;
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<TLink>
6      {
7          TLink Append(TLink sequence, TLink appendant);
8      }
9  }

```

./Sequences/ISequences.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.Sequences
7  {
8      public interface ISequences<TLink>
9      {
10         ulong Count(params TLink[] sequence);
11         bool Each(Func<TLink, bool> handler, IList<TLink> sequence);
12         bool EachPart(Func<TLink, bool> handler, TLink sequence);
13         TLink Create(params TLink[] sequence);
14         TLink Update(TLink[] sequence, TLink[] newSequence);
15         void Delete(params TLink[] sequence);
16     }
17 }

```

./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<TLink>
8      {
9          IEnumerable<IList<TLink>> Walk(TLink 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
    ↪ 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 закливания.
18  /// Альтернативой защиты от закливания 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  ///
23  /// TODO: Попробовать реализовать алгоритм используя Sigil (MSIL) и низкоуровневый стек и
    ↪ сравнить производительность.
24  /// </remarks>
25  public static class SequenceWalker
26  {
27      [MethodImpl(MethodImplOptions.AggressiveInlining)]
28      public static void WalkRight<TLink>(TLink sequence, Func<TLink, TLink> getSource,
    ↪ Func<TLink, TLink> getTarget, Func<TLink, bool> isElement, Action<TLink> visit)
29      {
30          var stack = new Stack<TLink>();
31          var element = sequence;
32          if (isElement(element))
33          {
34              visit(element);
35          }
36          else
37          {
38              while (true)
39              {
40                  if (isElement(element))
41                  {
42                      if (stack.Count == 0)
43                      {
44                          break;
45                      }
46                      element = stack.Pop();
47                      var source = getSource(element);
48                      var target = getTarget(element);
49                      if (isElement(source))
50                      {
51                          visit(source);
52                      }
53                      if (isElement(target))
54                      {
55                          visit(target);
56                      }
57                      element = target;
58                  }
59                  else
60                  {
61                      stack.Push(element);
62                      element = getSource(element);
63                  }
64              }
65          }
66      }
67
68      [MethodImpl(MethodImplOptions.AggressiveInlining)]
69      public static void WalkLeft<TLink>(TLink sequence, Func<TLink, TLink> getSource,
    ↪ Func<TLink, TLink> getTarget, Func<TLink, bool> isElement, Action<TLink> visit)
70      {
71          var stack = new Stack<TLink>();
72          var element = sequence;
73          if (isElement(element))
74          {
75              visit(element);
76          }
77          else
78          {
79              while (true)
80              {

```

```

81         if (isElement(element))
82         {
83             if (stack.Count == 0)
84             {
85                 break;
86             }
87             element = stack.Pop();
88             var source = getSource(element);
89             var target = getTarget(element);
90             if (isElement(target))
91             {
92                 visit(target);
93             }
94             if (isElement(source))
95             {
96                 visit(source);
97             }
98             element = source;
99         }
100     else
101     {
102         stack.Push(element);
103         element = getTarget(element);
104     }
105 }
106 }
107 }
108 }
109 }

```

./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
12     /// ↪ себя),
13     /// так как стек можно использовать намного эффективнее при ручном управлении.
14     ///
15     /// При оптимизации можно использовать встроенную поддержку стеков в процессор.
16     ///
17     /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
18     /// Решить встраивать ли защиту от заикливания.
19     /// Альтернативой защиты от заикливания может быть заранее известное ограничение на
20     /// ↪ погружение вглубь.
21     /// А так же качественное распознавание прохода по циклическому графу.
22     /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
23     /// ↪ стека.
24     /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
25     /// </remarks>
26     public static class StopableSequenceWalker
27     {
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public static bool WalkRight<TLink>(TLink sequence, Func<TLink, TLink> getSource,
30             ↪ Func<TLink, TLink> getTarget, Func<TLink, bool> isElement, Action<TLink> enter,
31             ↪ Action<TLink> exit, Func<TLink, bool> canEnter, Func<TLink, bool> visit)
32         {
33             var exited = 0;
34             var stack = new Stack<TLink>();
35             var element = sequence;
36             if (isElement(element))
37             {
38                 return visit(element);
39             }
40             while (true)
41             {
42                 if (isElement(element))
43                 {
44                     if (stack.Count == 0)
45                     {
46                         return true;
47                     }
48                     element = stack.Pop();
49                 }
50             }
51         }
52     }
53 }

```

```

44         exit(element);
45         exited++;
46         var source = getSource(element);
47         var target = getTarget(element);
48         if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
49             ↪ !visit(source))
50         {
51             return false;
52         }
53         if ((isElement(target) || !canEnter(target)) && !visit(target))
54         {
55             return false;
56         }
57         element = target;
58     }
59     else
60     {
61         if (canEnter(element))
62         {
63             enter(element);
64             exited = 0;
65             stack.Push(element);
66             element = getSource(element);
67         }
68         else
69         {
70             if (stack.Count == 0)
71             {
72                 return true;
73             }
74             element = stack.Pop();
75             exit(element);
76             exited++;
77             var source = getSource(element);
78             var target = getTarget(element);
79             if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
80                 ↪ !visit(source))
81             {
82                 return false;
83             }
84             if ((isElement(target) || !canEnter(target)) && !visit(target))
85             {
86                 return false;
87             }
88             element = target;
89         }
90     }
91 }
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 public static bool WalkRight<TLink>(TLink sequence, Func<TLink, TLink> getSource,
94 ↪ Func<TLink, TLink> getTarget, Func<TLink, bool> isElement, Func<TLink, bool> visit)
95 {
96     var stack = new Stack<TLink>();
97     var element = sequence;
98     if (isElement(element))
99     {
100         return visit(element);
101     }
102     while (true)
103     {
104         if (isElement(element))
105         {
106             if (stack.Count == 0)
107             {
108                 return true;
109             }
110             element = stack.Pop();
111             var source = getSource(element);
112             var target = getTarget(element);
113             if (isElement(source) && !visit(source))
114             {
115                 return false;
116             }
117             if (isElement(target) && !visit(target))
118             {
119                 return false;
120             }

```

```

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<TLink>(TLink sequence, Func<TLink, TLink> getSource,
132 ↪ Func<TLink, TLink> getTarget, Func<TLink, bool> isElement, Func<TLink, bool> visit)
133 {
134     var stack = new Stack<TLink>();
135     var element = sequence;
136     if (isElement(element))
137     {
138         return visit(element);
139     }
140     while (true)
141     {
142         if (isElement(element))
143         {
144             if (stack.Count == 0)
145             {
146                 return true;
147             }
148             element = stack.Pop();
149             var source = getSource(element);
150             var target = getTarget(element);
151             if (isElement(target) && !visit(target))
152             {
153                 return false;
154             }
155             if (isElement(source) && !visit(source))
156             {
157                 return false;
158             }
159             element = source;
160         }
161         else
162         {
163             stack.Push(element);
164             element = getTarget(element);
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<TLink>
12    {
13        TLink Read(ulong partType, TLink link);
14        bool Read(Func<TLink, bool> handler, params TLink[] pattern);
15        TLink Create(TLink[] parts);
16        TLink Update(TLink[] before, TLink[] after);
17        void Delete(TLink[] 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<TLink>
11    {
12        IList<IList<IList<TLink>>> Trigger(IList<TLink> condition, IList<TLink> substitution);
13    }
14
15    /// <remarks>Minimal sufficient universal Links API (for step by step operations).</remarks>
16    public partial interface IUniLinks<TLink>
17    {
18        /// <returns>
19        /// TLink that represents True (was finished fully) or TLink that represents False (was
20        ///   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        TLink Trigger(IList<TLink> patternOrCondition, Func<IList<TLink>, TLink> matchHandler,
29                    IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
30                    ↪ substitutionHandler);
31
32        TLink Trigger(IList<TLink> restriction, Func<IList<TLink>, IList<TLink>, TLink>
33                    ↪ matchedHandler,
34                    IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
35                    ↪ substitutedHandler);
36    }
37
38    /// <remarks>Extended with small optimization.</remarks>
39    public partial interface IUniLinks<TLink>
40    {
41        /// <remarks>
42        /// Something simple should be simple and optimized.
43        /// </remarks>
44        TLink Count(IList<TLink> restrictions);
45    }
46 }

```

./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<TLink>
12    {
13        TLink Get(ulong partType, TLink link);
14        bool Get(Func<TLink, bool> handler, params TLink[] pattern);
15        TLink Set(TLink[] before, TLink[] 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    /// TLink can be any number type of any size.
11    /// </remarks>
12    public interface IUniLinksIO<TLink>
13    {
14        /// <remarks>
15        /// default(TLink) 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<TLink[], bool> handler, params TLink[] pattern);
32
33     /// <remarks>
34     /// default(TLink) means itself.
35     /// Equivalent to:
36     /// * creation if before == null
37     /// * deletion if after == null
38     /// * update if before != null & & after != null
39     /// * default(TLink) 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>
    TLink In(TLink[] before, TLink[] after);
}
}

```

./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<TLink> : IUniLinksIO<TLink>
8      {
9          /// <remarks>
10         /// default(TLink) means nothing or null.
11         /// Single element pattern means just element (link).
12         /// OutPart(n, null) returns default(TLink).
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) ~ GetLinker(link) or GetLinker(Search(pattern))
17         /// OutPart(n, pattern) => For any variable length links, returns link or default(TLink).
18         ///
19         /// Outs(returns) inner contents of link, its part/parent/element/value.
20         /// </remarks>
21         TLink OutOne(ulong partType, params TLink[] pattern);
22
23         /// <remarks>OutCount() returns total links in store as array.</remarks>
24         TLink[][] OutAll(params TLink[] pattern);
25
26         /// <remarks>OutCount() returns total amount of links in store.</remarks>
27         ulong OutCount(params TLink[] pattern);
28     }
29 }

```

./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<TLink>
12 {
13     TLink Read(ulong partType, TLink link);
14     bool Read(Func<TLink, bool> handler, params TLink[] pattern);
15     TLink Write(TLink[] before, TLink[] after);
16 }
17 }
```


Index

- ./Constants/ILinksAddressConstants.cs, 1
- ./Constants/ILinksCombinedConstants.cs, 1
- ./Constants/ILinksDecisionConstants.cs, 1
- ./Constants/ILinksPartConstants.cs, 1
- ./Constants/LinksCombinedConstants[TDecision, TAddress, TPartIndex].cs, 2
- ./Constants/LinksCombinedConstants[TDecision, TAddress].cs, 2
- ./Constants/LinksDecisionConstants.cs, 2
- ./Exceptions/ArgumentLinkDoesNotExistsException.cs, 3
- ./Exceptions/ArgumentLinkHasDependenciesException.cs, 3
- ./Exceptions/LinkWithSameValueAlreadyExistsException.cs, 4
- ./Exceptions/LinksLimitReachedException.cs, 3
- ./ILinks.cs, 4
- ./ILinksExtensions.cs, 6
- ./ISynchronizedLinks.cs, 8
- ./Point.cs, 8
- ./Sequences/ISequenceAppender.cs, 9
- ./Sequences/ISequenceWalker.cs, 9
- ./Sequences/ISequences.cs, 9
- ./Sequences/SequenceWalker.cs, 9
- ./Sequences/StopableSequenceWalker.cs, 11
- ./Universal/IUniLinks.cs, 13
- ./Universal/IUniLinksCRUD.cs, 13
- ./Universal/IUniLinksGS.cs, 14
- ./Universal/IUniLinksIO.cs, 14
- ./Universal/IUniLinksIOWithExtensions.cs, 15
- ./Universal/IUniLinksRW.cs, 15