

LinksPlatform's Platform.Data.Doublets Class Library

1.1 ./csharp/Platform.Data.Doublets/CriterionMatchers/TargetMatcher.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.CriterionMatchers
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the target matcher.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="LinksOperatorBase{TLink}"/>
16    /// <seealso cref="ICriterionMatcher{TLink}"/>
17    public class TargetMatcher<TLink> : LinksOperatorBase<TLink>, ICriterionMatcher<TLink>
18    {
19        private static readonly EqualityComparer<TLink> _equalityComparer =
20            ↪ EqualityComparer<TLink>.Default;
21        private readonly TLink _targetToMatch;
22
23        /// <summary>
24        /// <para>
25        /// Initializes a new <see cref="TargetMatcher"/> instance.
26        /// </para>
27        /// <para></para>
28        /// </summary>
29        /// <param name="links">
30        /// <para>A links.</para>
31        /// </param>
32        /// <param name="targetToMatch">
33        /// <para>A target to match.</para>
34        /// </param>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public TargetMatcher(ILinks<TLink> links, TLink targetToMatch) : base(links) =>
38            ↪ _targetToMatch = targetToMatch;
39
40        /// <summary>
41        /// <para>
42        /// Determines whether this instance is matched.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="link">
47        /// <para>The link.</para>
48        /// </para>
49        /// </param>
50        /// <returns>
51        /// <para>The bool</para>
52        /// </para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        public bool IsMatched(TLink link) => _equalityComparer.Equals(_links.GetTarget(link),
56            ↪ _targetToMatch);
57    }
58 }
```

1.2 ./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Delegates;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Decorators
9 {
10    /// <summary>
11    /// <para>
12    /// Represents the links cascade uniqueness and usages resolver.
13    /// </para>
14    /// <para></para>
15    /// </summary>
16    /// <seealso cref="LinksUniquenessResolver{TLink}"/>
```

```

17 public class LinksCascadeUniquenessAndUsagesResolver<TLink> : LinksUniquenessResolver<TLink>
18 {
19     /// <summary>
20     /// <para>
21     /// Initializes a new <see cref="LinksCascadeUniquenessAndUsagesResolver"/> instance.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public LinksCascadeUniquenessAndUsagesResolver(ILinks<TLink> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Resolves the address change conflict using the specified old link address.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="oldLinkAddress">
39     /// <para>The old link address.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="newLinkAddress">
43     /// <para>The new link address.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The link</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
52     ↪ newLinkAddress, WriteHandler<TLink>? handler)
53     {
54         var constants = _links.Constants;
55         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
56         ↪ handler);
57         // Use Facade (the last decorator) to ensure recursion working correctly
58         handlerState.Apply(_facade.MergeUsages(oldLinkAddress, newLinkAddress,
59         ↪ handlerState.Handler));
60         handlerState.Apply(base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress,
61         ↪ handlerState.Handler));
62         return handlerState.Result;
63     }
64 }
65 }

```

1.3 ./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Delegates;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Decorators
9 {
10     /// <remarks>
11     /// <para>Must be used in conjunction with NonNullContentsLinkDeletionResolver.</para>
12     /// <para>Должен использоваться вместе с NonNullContentsLinkDeletionResolver.</para>
13     /// </remarks>
14     public class LinksCascadeUsagesResolver<TLink> : LinksDecoratorBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksCascadeUsagesResolver"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="links">
23         /// <para>A links.</para>
24         /// <para></para>
25         /// </param>
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }

```

```

28
29     /// <summary>
30     /// <para>
31     /// Deletes the restriction.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="restriction">
36     /// <para>The restriction.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public override TLink Delete(ICollection<TLink>? restriction, WriteHandler<TLink> handler)
41     {
42         var constants = _links.Constants;
43         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
44             ↪ handler);
45         var linkIndex = restriction[_constants.IndexPart];
46         // Use Facade (the last decorator) to ensure recursion working correctly
47         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
48         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
49         return handlerState.Result;
50     }
51 }

```

1.4 ./csharp/Platform.Data.Doublets/Decorators/LinksDecoratorBase.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links decorator base.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}" />
17     /// <seealso cref="ILinks{TLink}" />
18     public abstract class LinksDecoratorBase<TLink> : LinksOperatorBase<TLink>, ILinks<TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// The constants.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         protected readonly LinksConstants<TLink> _constants;
27
28         /// <summary>
29         /// <para>
30         /// Gets the constants value.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         public LinksConstants<TLink> Constants
35         {
36             [MethodImpl(MethodImplOptions.AggressiveInlining)]
37             get => _constants;
38         }
39
40         /// <summary>
41         /// <para>
42         /// The facade.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         protected ILinks<TLink> _facade;
47
48         /// <summary>
49         /// <para>
50         /// Gets or sets the facade value.
51         /// </para>
52         /// <para></para>

```

```

53     /// </summary>
54     public ILinks<TLink> Facade
55     {
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         get => _facade;
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         set
60         {
61             _facade = value;
62             if (_links is LinksDecoratorBase<TLink> decorator)
63             {
64                 decorator.Facade = value;
65             }
66         }
67     }
68
69     /// <summary>
70     /// <para>
71     /// Initializes a new <see cref="LinksDecoratorBase"/> instance.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <param name="links">
76     /// <para>A links.</para>
77     /// <para></para>
78     /// </param>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected LinksDecoratorBase(ILinks<TLink> links) : base(links)
81     {
82         _constants = links.Constants;
83         Facade = this;
84     }
85
86     /// <summary>
87     /// <para>
88     /// Counts the restriction.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="restriction">
93     /// <para>The restriction.</para>
94     /// <para></para>
95     /// </param>
96     /// <returns>
97     /// <para>The link</para>
98     /// <para></para>
99     /// </returns>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    public virtual TLink Count(IList<TLink>? restriction) => _links.Count(restriction);
102
103    /// <summary>
104    /// <para>
105    /// Eaches the handler.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    /// <param name="handler">
110    /// <para>The handler.</para>
111    /// <para></para>
112    /// </param>
113    /// <param name="restriction">
114    /// <para>The restriction.</para>
115    /// <para></para>
116    /// </param>
117    /// <returns>
118    /// <para>The link</para>
119    /// <para></para>
120    /// </returns>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    public virtual TLink Each(IList<TLink> restriction, ReadHandler<TLink>? handler) =>
123        ↪ _links.Each(restriction, handler);
124
125    /// <summary>
126    /// <para>
127    /// Creates the restriction.
128    /// </para>
129    /// <para></para>
130    /// </summary>

```

```

130     /// <param name="restriction">
131     /// <para>The restriction.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The link</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public virtual TLink Create(IList<TLink> substitution, WriteHandler<TLink> handler) =>
140         ↪ _links.Create(substitution, handler);
141
142     /// <summary>
143     /// <para>
144     /// Updates the restriction.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="restriction">
149     /// <para>The restriction.</para>
150     /// <para></para>
151     /// </param>
152     /// <param name="substitution">
153     /// <para>The substitution.</para>
154     /// <para></para>
155     /// </param>
156     /// <returns>
157     /// <para>The link</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public virtual TLink Update(IList<TLink> restriction, IList<TLink> substitution,
162         ↪ WriteHandler<TLink> handler) => _links.Update(restriction, substitution, handler);
163
164     /// <summary>
165     /// <para>
166     /// Deletes the restriction.
167     /// </para>
168     /// <para></para>
169     /// </summary>
170     /// <param name="restriction">
171     /// <para>The restriction.</para>
172     /// <para></para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     public virtual TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler) =>
176         ↪ _links.Delete(restriction, handler);
177 }
178 }

```

1.5 ./csharp/Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Disposables;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5 #pragma warning disable CA1063 // Implement IDisposable Correctly
6
7 namespace Platform.Data.Doublets.Decorators
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the links disposable decorator base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksDecoratorBase{TLink}">
16     /// <seealso cref="ILinks{TLink}">
17     /// <seealso cref="System.IDisposable">
18     public abstract class LinksDisposableDecoratorBase<TLink> : LinksDecoratorBase<TLink>,
19         ↪ ILinks<TLink>, System.IDisposable
20     {
21         /// <summary>
22         /// <para>
23         /// Represents the disposable with multiple calls allowed.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <seealso cref="Disposable">

```

```

27 protected class DisposableWithMultipleCallsAllowed : Disposable
28 {
29     /// <summary>
30     /// <para>
31     /// Initializes a new <see cref="DisposableWithMultipleCallsAllowed"/> instance.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="disposal">
36     /// <para>A disposal.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public DisposableWithMultipleCallsAllowed(Disposal disposal) : base(disposal) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the allow multiple dispose calls value.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     protected override bool AllowMultipleDisposeCalls
49     {
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         get => true;
52     }
53 }
54
55 /// <summary>
56 /// <para>
57 /// The disposable.
58 /// </para>
59 /// <para></para>
60 /// </summary>
61 protected readonly DisposableWithMultipleCallsAllowed Disposable;
62
63 /// <summary>
64 /// <para>
65 /// Initializes a new <see cref="LinksDisposableDecoratorBase"/> instance.
66 /// </para>
67 /// <para></para>
68 /// </summary>
69 /// <param name="links">
70 /// <para>A links.</para>
71 /// <para></para>
72 /// </param>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 protected LinksDisposableDecoratorBase(ILinks<TLink> links) : base(links) => Disposable
75     ↳ = new DisposableWithMultipleCallsAllowed(Dispose);
76
77 [MethodImpl(MethodImplOptions.AggressiveInlining)]
78 ~LinksDisposableDecoratorBase() => Disposable.Destruct();
79
80 /// <summary>
81 /// <para>
82 /// Disposes this instance.
83 /// </para>
84 /// <para></para>
85 /// </summary>
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 public void Dispose() => Disposable.Dispose();
88
89 /// <summary>
90 /// <para>
91 /// Disposes the manual.
92 /// </para>
93 /// <para></para>
94 /// </summary>
95 /// <param name="manual">
96 /// <para>The manual.</para>
97 /// <para></para>
98 /// </param>
99 /// <param name="wasDisposed">
100 /// <para>The was disposed.</para>
101 /// <para></para>
102 /// </param>
103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
104 protected virtual void Dispose(bool manual, bool wasDisposed)

```

```

104     {
105         if (!wasDisposed)
106         {
107             _links.DisposeIfPossible();
108         }
109     }
110 }
111 }

```

1.6 ./csharp/Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     // TODO: Make LinksExternalReferenceValidator. A layer that checks each link to exist or to
11     // ↳ be external (hybrid link's raw number).
12     /// <summary>
13     /// <para>
14     /// Represents the links inner reference existence validator.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="LinksDecoratorBase{TLink}" />
19     public class LinksInnerReferenceExistenceValidator<TLink> : LinksDecoratorBase<TLink>
20     {
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="LinksInnerReferenceExistenceValidator" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public LinksInnerReferenceExistenceValidator(ILinks<TLink> links) : base(links) { }
33
34         /// <summary>
35         /// <para>
36         /// Eaches the handler.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="handler">
41         /// <para>The handler.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="restriction">
45         /// <para>The restriction.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The link</para>
50         /// <para></para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         public override TLink Each(IList<TLink>? restriction, ReadHandler<TLink>? handler)
54         {
55             var links = _links;
56             links.EnsureInnerReferenceExists(restriction, nameof(restriction));
57             return links.Each(restriction, handler);
58         }
59
60         /// <summary>
61         /// <para>
62         /// Updates the restriction.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <param name="restriction">
67         /// <para>The restriction.</para>
68         /// <para></para>
69         /// </param>

```

```

69     /// <param name="substitution">
70     /// <para>The substitution.</para>
71     /// <para></para>
72     /// </param>
73     /// <returns>
74     /// <para>The link</para>
75     /// <para></para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public override TLink Update(IList<TLink> restriction, IList<TLink> substitution,
    ↪ WriteHandler<TLink> handler)
79     {
80         // TODO: Possible values: null, ExistentLink or NonExistentHybrid(ExternalReference)
81         var links = _links;
82         links.EnsureInnerReferenceExists(restriction, nameof(restriction));
83         links.EnsureInnerReferenceExists(substitution, nameof(substitution));
84         return links.Update(restriction, substitution, handler);
85     }
86
87     /// <summary>
88     /// <para>
89     /// Deletes the restriction.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="restriction">
94     /// <para>The restriction.</para>
95     /// <para></para>
96     /// </param>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     public override TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler)
99     {
100         var link = restriction[_constants.IndexPart];
101         var links = _links;
102         links.EnsureLinkExists(link, nameof(link));
103         return links.Delete(restriction, handler);
104     }
105 }
106 }

```

1.7 ./csharp/Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links itself constant to self reference resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase<TLink>" />
17     public class LinksItselfConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="LinksItselfConstantToSelfReferenceResolver" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public LinksItselfConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }
33
34         /// <summary>
35         /// <para>
36         /// Eaches the handler.
37         /// </para>

```



```

38     /// <para></para>
39     /// </summary>
40     /// <param name="handler">
41     /// <para>The handler.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="restriction">
45     /// <para>The restriction.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public override TLink Each(IList<TLink>? restriction, ReadHandler<TLink>? handler)
54     {
55         var constants = _constants;
56         var itselfConstant = constants.Itself;
57         if (!_equalityComparer.Equals(constants.Any, itselfConstant) &&
58             ↪ restriction.Contains(itselfConstant))
59         {
60             // Itself constant is not supported for Each method right now, skipping execution
61             return constants.Continue;
62         }
63         return _links.Each(restriction, handler);
64     }
65     /// <summary>
66     /// <para>
67     /// Updates the restriction.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="restriction">
72     /// <para>The restriction.</para>
73     /// <para></para>
74     /// </param>
75     /// <param name="substitution">
76     /// <para>The substitution.</para>
77     /// <para></para>
78     /// </param>
79     /// <returns>
80     /// <para>The link</para>
81     /// <para></para>
82     /// </returns>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     public override TLink Update(IList<TLink> restriction, IList<TLink> substitution,
85         ↪ WriteHandler<TLink> handler) => _links.Update(restriction,
86         ↪ _links.ResolveConstantAsSelfReference(_constants.Itself, restriction, substitution),
87         ↪ handler);
88     }
89 }

```

1.8 ./csharp/Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <remarks>
11     /// Not practical if newSource and newTarget are too big.
12     /// To be able to use practical version we should allow to create link at any specific
13     ↪ location inside ResizableDirectMemoryLinks.
14     /// This in turn will require to implement not a list of empty links, but a list of ranges
15     ↪ to store it more efficiently.
16     /// </remarks>
17     public class LinksNonExistentDependenciesCreator<TLink> : LinksDecoratorBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LinksNonExistentDependenciesCreator"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>

```

```

23     /// <param name="links">
24     /// <para>A links.</para>
25     /// <para></para>
26     /// </param>
27     [MethodImpl(MethodImplOptions.AggressiveInlining)]
28     public LinksNonExistentDependenciesCreator(ILinks<TLink> links) : base(links) { }
29
30     /// <summary>
31     /// <para>
32     /// Updates the restriction.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     /// <param name="restriction">
37     /// <para>The restriction.</para>
38     /// <para></para>
39     /// </param>
40     /// <param name="substitution">
41     /// <para>The substitution.</para>
42     /// <para></para>
43     /// </param>
44     /// <returns>
45     /// <para>The link</para>
46     /// <para></para>
47     /// </returns>
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     public override TLink Update(IList<TLink>? restriction, IList<TLink> substitution,
50         ↳ WriteHandler<TLink> handler)
51     {
52         var constants = _constants;
53         var links = _links;
54         links.EnsureCreated(substitution[constants.SourcePart],
55             ↳ substitution[constants.TargetPart]);
56         return links.Update(restriction, substitution, handler);
57     }
58 }

```

1.9 ./csharp/Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links null constant to self reference resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}" />
17     public class LinksNullConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LinksNullConstantToSelfReferenceResolver" /> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="links">
26         /// <para>A links.</para>
27         /// <para></para>
28         /// </param>
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public LinksNullConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }
31
32         /// <summary>
33         /// <para>
34         /// Creates the substitution.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="substitution">
39         /// <para>The substitution.</para>
40         /// <para></para>

```

```

41     /// </param>
42     /// <returns>
43     /// <para>The link</para>
44     /// <para></para>
45     /// </returns>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public override TLink Create(ICollection<TLink>? substitution, WriteHandler<TLink> handler)
48     {
49         return _links.CreatePoint(handler);
50     }
51
52     /// <summary>
53     /// <para>
54     /// Updates the substitution.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="restriction">
59     /// <para>The substitution.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="substitution">
63     /// <para>The substitution.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public override TLink Update(ICollection<TLink> restriction, ICollection<TLink> substitution,
72         WriteHandler<TLink> handler) => _links.Update(restriction,
73         _links.ResolveConstantAsSelfReference(_constants.Null, restriction, substitution),
74         handler);
75 }

```

1.10 ./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessResolver.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links uniqueness resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}" />
17     public class LinksUniquenessResolver<TLink> : LinksDecoratorBase<TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
20             EqualityComparer<TLink>.Default;
21
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="LinksUniquenessResolver" /> instance.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public LinksUniquenessResolver(ICollection<TLink> links) : base(links) { }
34
35         /// <summary>
36         /// <para>
37         /// Updates the restriction.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="restriction">

```

```

41     /// <para>The restriction.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="substitution">
45     /// <para>The substitution.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public override TLink Update(IList<TLink>? restriction, IList<TLink> substitution,
54     ↪ WriteHandler<TLink> handler)
55     {
56         var constants = _constants;
57         var links = _links;
58         var newLinkAddress = links.SearchOrDefault(substitution[constants.SourcePart],
59     ↪ substitution[constants.TargetPart]);
60         if (_equalityComparer.Equals(newLinkAddress, default))
61         {
62             return links.Update(restriction, substitution, handler);
63         }
64         return ResolveAddressChangeConflict(restriction[constants.IndexPart],
65     ↪ newLinkAddress, handler);
66     }
67
68     /// <summary>
69     /// <para>
70     /// Resolves the address change conflict using the specified old link address.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="oldLinkAddress">
75     /// <para>The old link address.</para>
76     /// <para></para>
77     /// </param>
78     /// <param name="newLinkAddress">
79     /// <para>The new link address.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The new link address.</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected virtual TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
88     ↪ newLinkAddress, WriteHandler<TLink> handler)
89     {
90         if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
91     ↪ _links.Exists(oldLinkAddress))
92         {
93             return _facade.Delete(oldLinkAddress, handler);
94         }
95         return _links.Constants.Continue;
96     }
97 }
98
99 }

```

1.11 ./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessValidator.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links uniqueness validator.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}">
17     public class LinksUniquenessValidator<TLink> : LinksDecoratorBase<TLink>
18     {

```

```

19     /// <summary>
20     /// <para>
21     /// Initializes a new <see cref="LinksUniquenessValidator"/> instance.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public LinksUniquenessValidator(ILinks<TLink> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Updates the restriction.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="restriction">
39     /// <para>The restriction.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="substitution">
43     /// <para>The substitution.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The link</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public override TLink Update(IList<TLink>? restriction, IList<TLink> substitution,
52     ↪ WriteHandler<TLink> handler)
53     {
54         var links = _links;
55         var constants = _constants;
56         links.EnsureDoesNotExists(substitution[constants.SourcePart],
57         ↪ substitution[constants.TargetPart]);
58         return links.Update(restriction, substitution, handler);
59     }

```

1.12 ./csharp/Platform.Data.Doublets/Decorators/LinksUsagesValidator.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links usages validator.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}"/>
17     public class LinksUsagesValidator<TLink> : LinksDecoratorBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LinksUsagesValidator"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="links">
26         /// <para>A links.</para>
27         /// <para></para>
28         /// </param>
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public LinksUsagesValidator(ILinks<TLink> links) : base(links) { }
31
32         /// <summary>
33         /// <para>
34         /// Updates the restriction.

```

```

35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="restriction">
39     /// <para>The restriction.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="substitution">
43     /// <para>The substitution.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The link</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public override TLink Update(ICollection<TLink>? restriction, ICollection<TLink> substitution,
    ↪ WriteHandler<TLink> handler)
52     {
53         var links = _links;
54         links.EnsureNoUsages(restriction[_constants.IndexPart]);
55         return links.Update(restriction, substitution, handler);
56     }
57
58     /// <summary>
59     /// <para>
60     /// Deletes the restriction.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="restriction">
65     /// <para>The restriction.</para>
66     /// <para></para>
67     /// </param>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public override TLink Delete(ICollection<TLink> restriction, WriteHandler<TLink> handler)
70     {
71         var link = restriction[_constants.IndexPart];
72         var links = _links;
73         links.EnsureNoUsages(link);
74         return links.Delete(restriction, handler);
75     }
76 }
77 }

```

1.13 ./csharp/Platform.Data.Doublets/Decorators/LoggingDecorator.cs

```

1  using System.Collections.Generic;
2  using System.IO;
3  using Platform.Delegates;
4
5  namespace Platform.Data.Doublets.Decorators
6  {
7      public class LoggingDecorator<TLink> : LinksDecoratorBase<TLink>
8      {
9          private readonly Stream _logStream;
10         private readonly StreamWriter _logStreamWriter;
11         public LoggingDecorator(ICollection<TLink> links, Stream logStream) : base(links)
12         {
13             _logStream = logStream;
14             _logStreamWriter = new StreamWriter(_logStream);
15             _logStreamWriter.AutoFlush = true;
16         }
17
18         public override TLink Create(ICollection<TLink>? substitution, WriteHandler<TLink> handler)
19         {
20             WriteHandlerState<TLink> handlerState = new(_constants.Continue, _constants.Break,
    ↪ handler);
21             return base.Create(substitution, (before, after) =>
22             {
23                 if (handlerState.Handler != null)
24                 {
25                     handlerState.Apply(handlerState.Handler(before, after));
26                 }
27                 _logStreamWriter.WriteLine($"Create. Before: {new Link<TLink>(before)}. After:
    ↪ {new Link<TLink>(after)}");
28                 return _constants.Continue;
29             });
30         }
31     }

```

```

32     public override TLink Update(IList<TLink> restriction, IList<TLink> substitution,
    ↪ WriteHandler<TLink> handler)
33     {
34         WriteHandlerState<TLink> handlerState = new(_constants.Continue, _constants.Break,
    ↪ handler);
35         return base.Update(restriction, substitution, (before, after) =>
36         {
37             if (handlerState.Handler != null)
38             {
39                 handlerState.Apply(handlerState.Handler(before, after));
40             }
41             _logStreamWriter.WriteLine($"Update. Before: {new Link<TLink>(before)}. After:
    ↪ {new Link<TLink>(after)}");
42             return _constants.Continue;
43         });
44     }
45
46     public override TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler)
47     {
48         WriteHandlerState<TLink> handlerState = new(_constants.Continue, _constants.Break,
    ↪ handler);
49         return base.Delete(restriction, (before, after) =>
50         {
51             if (handlerState.Handler != null)
52             {
53                 handlerState.Apply(handlerState.Handler(before, after));
54             }
55             _logStreamWriter.WriteLine($"Delete. Before: {new Link<TLink>(before)}. After:
    ↪ {new Link<TLink>(after)}");
56             return _constants.Continue;
57         });
58     }
59 }
60 }

```

1.14 ./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the non null contents link deletion resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}">
17     public class NonNullContentsLinkDeletionResolver<TLink> : LinksDecoratorBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="NonNullContentsLinkDeletionResolver"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="links">
26         /// <para>A links.</para>
27         /// <para></para>
28         /// </param>
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public NonNullContentsLinkDeletionResolver(ILinks<TLink> links) : base(links) { }
31
32         /// <summary>
33         /// <para>
34         /// Deletes the restriction.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="restriction">
39         /// <para>The restriction.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

43     public override TLink Delete(IList<TLink>? restriction, WriteHandler<TLink> handler)
44     {
45         var linkIndex = restriction[_constants.IndexPart];
46         var constants = _links.Constants;
47         WriteHandlerState<TLink> handlerResult = new(constants.Continue, constants.Break,
48             ↪ handler);
49         handlerResult.Apply(_links.EnforceResetValues(linkIndex, handlerResult.Handler));
50         handlerResult.Apply(_links.Delete(restriction, handlerResult.Handler));
51         return handlerResult.Result;
52     }
53 }

```

1.15 ./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5  using TLink = System.UInt32;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Decorators
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 32 links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksDisposableDecoratorBase{TLink}"/>
18     public class UInt32Links : LinksDisposableDecoratorBase<TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32Links"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public UInt32Links(ILinks<TLink> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Creates the substitution.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="substitution">
40         /// <para>The substitution.</para>
41         /// <para></para>
42         /// </param>
43         /// <returns>
44         /// <para>The link</para>
45         /// <para></para>
46         /// </returns>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public override TLink Create(IList<TLink>? substitution, WriteHandler<TLink> handler) =>
49             ↪ _links.CreatePoint(handler);
50
51         /// <summary>
52         /// <para>
53         /// Updates the substitution.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         /// <param name="restriction">
58         /// <para>The substitution.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="substitution">
62         /// <para>The substitution.</para>
63         /// <para></para>
64         /// </param>
65         /// <returns>

```



```

65     /// <para>The link</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public override TLink Update(ICollection<TLink> restriction, ICollection<TLink> substitution,
70     ↪ WriteHandler<TLink> handler)
71     {
72         var constants = _constants;
73         var indexPartConstant = constants.IndexPart;
74         var sourcePartConstant = constants.SourcePart;
75         var targetPartConstant = constants.TargetPart;
76         var nullConstant = constants.Null;
77         var itselfConstant = constants.Itself;
78         var existedLink = nullConstant;
79         var updatedLink = restriction[indexPartConstant];
80         var newSource = substitution[sourcePartConstant];
81         var newTarget = substitution[targetPartConstant];
82         var links = _links;
83         if (newSource != itselfConstant && newTarget != itselfConstant)
84         {
85             existedLink = links.SearchOrDefault(newSource, newTarget);
86         }
87         if (existedLink == nullConstant)
88         {
89             var before = links.GetLink(updatedLink);
90             if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
91             ↪ newTarget)
92             {
93                 var source = newSource == itselfConstant ? updatedLink : newSource;
94                 var target = newTarget == itselfConstant ? updatedLink : newTarget;
95                 return links.Update(new Link<TLink>(updatedLink, source, target), handler);
96             }
97             return _links.Constants.Continue;
98         }
99         else
100         {
101             return _facade.MergeAndDelete(updatedLink, existedLink, handler);
102         }
103     }
104     /// <summary>
105     /// <para>
106     /// Deletes the substitution.
107     /// </para>
108     /// </summary>
109     /// <param name="restriction">
110     /// <para>The restriction.</para>
111     /// </param>
112     /// </summary>
113     [MethodImpl(MethodImplOptions.AggressiveInlining)]
114     public override TLink Delete(ICollection<TLink> restriction, WriteHandler<TLink> handler)
115     {
116         var linkIndex = restriction[_constants.IndexPart];
117         var constants = _links.Constants;
118         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
119         ↪ handler);
120         handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));
121         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
122         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
123         return handlerState.Result;
124     }
125 }

```

1.16 ./csharp/Platform.Data.Doublets.Decorators/UInt64Links.cs

```

1  using System.Collections.Generic;
2  using System.Net.Security;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5  using TLink = System.UInt64;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Decorators
10 {
11     /// <summary>
12     /// <para>Represents a combined decorator that implements the basic logic for interacting
13     ↪ with the links storage for links with addresses represented as <see cref="System.UInt64"
14     ↪ />.</para>

```

```

13  /// <para>Представляет комбинированный декоратор, реализующий основную логику по
    ↳ взаимодействию с хранилищем связей, для связей с адресами представленными в виде <see
    ↳ cref="System.UInt64"/>.</para>
14  /// </summary>
15  /// <remarks>
16  /// Возможные оптимизации:
17  /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
18  ///     + меньше объём БД
19  ///     - меньше производительность
20  ///     - больше ограничение на количество связей в БД)
21  /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
22  ///     + меньше объём БД
23  ///     - больше сложность
24  ///
25  /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
    ↳ поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
    ↳ 460 752 303 423 488
26  /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
    ↳ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
27  ///
28  /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
    ↳ выбрасываться только при #if DEBUG
29  /// </remarks>
30  public class UInt64Links : LinksDisposableDecoratorBase<TLink>
31  {
32      /// <summary>
33      /// <para>
34      /// Initializes a new <see cref="UInt64Links"/> instance.
35      /// </para>
36      /// <para></para>
37      /// </summary>
38      /// <param name="links">
39      /// <para>A links.</para>
40      /// <para></para>
41      /// </param>
42      [MethodImpl(MethodImplOptions.AggressiveInlining)]
43      public UInt64Links(ILinks<TLink> links) : base(links) { }
44
45      /// <summary>
46      /// <para>
47      /// Creates the substitution.
48      /// </para>
49      /// <para></para>
50      /// </summary>
51      /// <param name="substitution">
52      /// <para>The substitution.</para>
53      /// <para></para>
54      /// </param>
55      /// <returns>
56      /// <para>The TLink</para>
57      /// <para></para>
58      /// </returns>
59      [MethodImpl(MethodImplOptions.AggressiveInlining)]
60      public override TLink Create(IList<TLink>? substitution, WriteHandler<TLink> handler) =>
        ↳ _links.CreatePoint(handler);
61
62      /// <summary>
63      /// <para>
64      /// Updates the substitution.
65      /// </para>
66      /// <para></para>
67      /// </summary>
68      /// <param name="restriction">
69      /// <para>The substitution.</para>
70      /// <para></para>
71      /// </param>
72      /// <param name="substitution">
73      /// <para>The substitution.</para>
74      /// <para></para>
75      /// </param>
76      /// <returns>
77      /// <para>The TLink</para>
78      /// <para></para>
79      /// </returns>
80      [MethodImpl(MethodImplOptions.AggressiveInlining)]
81      public override TLink Update(IList<TLink> restriction, IList<TLink> substitution,
        ↳ WriteHandler<TLink> handler)

```

```

82     {
83         var constants = _constants;
84         var indexPartConstant = constants.IndexPart;
85         var sourcePartConstant = constants.SourcePart;
86         var targetPartConstant = constants.TargetPart;
87         var nullConstant = constants.Null;
88         var itselfConstant = constants.Itself;
89         var existedLink = nullConstant;
90         var updatedLink = restriction[indexPartConstant];
91         var newSource = substitution[sourcePartConstant];
92         var newTarget = substitution[targetPartConstant];
93         var links = _links;
94         if (newSource != itselfConstant && newTarget != itselfConstant)
95         {
96             existedLink = links.SearchOrDefault(newSource, newTarget);
97         }
98         if (existedLink == nullConstant)
99         {
100             var before = links.GetLink(updatedLink);
101             if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
102                 ↪ newTarget)
103             {
104                 var source = newSource == itselfConstant ? updatedLink : newSource;
105                 var target = newTarget == itselfConstant ? updatedLink : newTarget;
106                 return links.Update(new Link<TLink>(updatedLink, source, target), handler);
107             }
108             return _links.Constants.Continue;
109         }
110         else
111         {
112             return _facade.MergeAndDelete(updatedLink, existedLink, handler);
113         }
114     }
115     /// <summary>
116     /// <para>
117     /// Deletes the substitution.
118     /// </para>
119     /// <para></para>
120     /// </summary>
121     /// <param name="restriction">
122     /// <para>The substitution.</para>
123     /// <para></para>
124     /// </param>
125     [MethodImpl(MethodImplOptions.AggressiveInlining)]
126     public override TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler)
127     {
128         var linkIndex = restriction[_constants.IndexPart];
129         var constants = _links.Constants;
130         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
131             ↪ handler);
132         handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));
133         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
134         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
135         return handlerState.Result;
136     }
137 }

```

1.17 ./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Platform.Collections;
5  using Platform.Collections.Lists;
6  using Platform.Data.Universal;
7  using Platform.Delegates;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Decorators
12 {
13     /// <remarks>
14     /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
15     /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
16     ↪ by itself. But can cause creation (update from nothing) or deletion (update to nothing).
17     ///
18     /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
19     ↪ DefaultUniLinksBase, that contains logic itself and can be implemented using both
20     ↪ IDoubletLinks and ILinks.)

```

```

18  /// </remarks>
19  internal class UniLinks<TLink> : LinksDecoratorBase<TLink>, IUniLinks<TLink>
20  {
21      private static readonly EqualityComparer<TLink> _equalityComparer =
22          ↳ EqualityComparer<TLink>.Default;
23
24      /// <summary>
25      /// <para>
26      /// Initializes a new <see cref="UniLinks"/> instance.
27      /// </para>
28      /// </summary>
29      /// <param name="links">
30      /// <para>A links.</para>
31      /// </param>
32      public UniLinks(ILinks<TLink> links) : base(links) { }
33      private struct Transition
34      {
35          /// <summary>
36          /// <para>
37          /// The before.
38          /// </para>
39          /// </summary>
40          public IList<TLink>? Before;
41          /// <summary>
42          /// <para>
43          /// The after.
44          /// </para>
45          /// </summary>
46          public IList<TLink> After;
47
48          /// <summary>
49          /// <para>
50          /// Initializes a new <see cref="Transition"/> instance.
51          /// </para>
52          /// </summary>
53          /// <param name="before">
54          /// <para>A before.</para>
55          /// </param>
56          /// <param name="after">
57          /// <para>A after.</para>
58          /// </param>
59          public Transition(IList<TLink> before, IList<TLink> after)
60          {
61              Before = before;
62              After = after;
63          }
64      }
65
66      //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
67      //public static readonly IReadOnlyList<TLink> NullLink = new
68      ↳ ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
69      ↳ });
70
71      // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
72      ↳ (Links-Expression)
73      /// <summary>
74      /// <para>
75      /// Triggers the restriction.
76      /// </para>
77      /// </summary>
78      /// <param name="restriction">
79      /// <para>The restriction.</para>
80      /// </param>
81      /// <param name="matchedHandler">
82      /// <para>The matched handler.</para>
83      /// </param>
84      /// <param name="substitution">
85      /// <para>The substitution.</para>

```

```

92  /// <para></para>
93  /// </param>
94  /// <param name="substitutedHandler">
95  /// <para>The substituted handler.</para>
96  /// <para></para>
97  /// </param>
98  /// <returns>
99  /// <para>The link</para>
100 /// <para></para>
101 /// </returns>
102 public TLink Trigger(IList<TLink> restriction, WriteHandler<TLink> matchedHandler,
    ↳ IList<TLink> substitution, WriteHandler<TLink> substitutedHandler)
103 {
104     ///List<Transition> transitions = null;
105     ///if (!restriction.IsNullOrEmpty())
106     ///{
107     ///    // Есть причина делать проход (чтение)
108     ///    if (matchedHandler != null)
109     ///    {
110     ///        if (!substitution.IsNullOrEmpty())
111     ///        {
112     ///            // restriction => { 0, 0, 0 } | { 0 } // Create
113     ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
    ↳ Create / Update
114     ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
115     ///            transitions = new List<Transition>();
116     ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
117     ///            {
118     ///                // If index is Null, that means we always ignore every other
    ↳ value (they are also Null by definition)
119     ///                var matchDecision = matchedHandler(, NullLink);
120     ///                if (Equals(matchDecision, Constants.Break))
121     ///                    return false;
122     ///                if (!Equals(matchDecision, Constants.Skip))
123     ///                    transitions.Add(new Transition(matchedLink, newValue));
124     ///            }
125     ///            else
126     ///            {
127     ///                Func<T, bool> handler;
128     ///                handler = link =>
129     ///                {
130     ///                    var matchedLink = Memory.GetLinkValue(link);
131     ///                    var newValue = Memory.GetLinkValue(link);
132     ///                    newValue[Constants.IndexPart] = Constants.Itself;
133     ///                    newValue[Constants.SourcePart] =
    ↳ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
134     ///                    newValue[Constants.TargetPart] =
    ↳ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
135     ///                    var matchDecision = matchedHandler(matchedLink, newValue);
136     ///                    if (Equals(matchDecision, Constants.Break))
137     ///                        return false;
138     ///                    if (!Equals(matchDecision, Constants.Skip))
139     ///                        transitions.Add(new Transition(matchedLink, newValue));
140     ///                    return true;
141     ///                };
142     ///                if (!Memory.Each(handler, restriction))
143     ///                    return Constants.Break;
144     ///            }
145     ///        }
146     ///    }
147     ///    else
148     ///    {
149     ///        Func<T, bool> handler = link =>
150     ///        {
151     ///            var matchedLink = Memory.GetLinkValue(link);
152     ///            var matchDecision = matchedHandler(matchedLink, matchedLink);
153     ///            return !Equals(matchDecision, Constants.Break);
154     ///        };
155     ///        if (!Memory.Each(handler, restriction))
156     ///            return Constants.Break;
157     ///    }
158     ///    else
159     ///    {
160     ///        if (substitution != null)
161     ///        {

```

```

162         transitions = new List<ILink<T>>>();
163         Func<T, bool> handler = link =>
164         {
165             var matchedLink = Memory.GetLinkValue(link);
166             transitions.Add(matchedLink);
167             return true;
168         };
169         if (!Memory.Each(handler, restriction))
170             return Constants.Break;
171     }
172     else
173     {
174         return Constants.Continue;
175     }
176 }
177 }
178 }
179 if (substitution != null)
180 {
181     // Есть причина делать замену (запись)
182     if (substitutedHandler != null)
183     {
184     }
185     else
186     {
187     }
188 }
189 return Constants.Continue;
190
191 //if (restriction.IsNullOrEmpty()) // Create
192 //{
193     substitution[Constants.IndexPart] = Memory.AllocateLink();
194     Memory.SetLinkValue(substitution);
195 //}
196 //else if (substitution.IsNullOrEmpty()) // Delete
197 //{
198     Memory.FreeLink(restriction[Constants.IndexPart]);
199 //}
200 //else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
201 //{
202     // No need to collect links to list
203     // Skip == Continue
204     // No need to check substitutedHandler
205     if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
206         ↪ Constants.Break), restriction))
207         return Constants.Break;
208 //}
209 //else // Update
210 //{
211     //List<ILink<T>> matchedLinks = null;
212     if (matchedHandler != null)
213     {
214         matchedLinks = new List<ILink<T>>>();
215         Func<T, bool> handler = link =>
216         {
217             var matchedLink = Memory.GetLinkValue(link);
218             var matchDecision = matchedHandler(matchedLink);
219             if (Equals(matchDecision, Constants.Break))
220                 return false;
221             if (!Equals(matchDecision, Constants.Skip))
222                 matchedLinks.Add(matchedLink);
223             return true;
224         };
225         if (!Memory.Each(handler, restriction))
226             return Constants.Break;
227     }
228     if (!matchedLinks.IsNullOrEmpty())
229     {
230         var totalMatchedLinks = matchedLinks.Count;
231         for (var i = 0; i < totalMatchedLinks; i++)
232         {
233             var matchedLink = matchedLinks[i];
234             if (substitutedHandler != null)
235             {
236                 var newValue = new List<T>(); // TODO: Prepare value to update here
237                 // TODO: Decide is it actually needed to use Before and After
238                 ↪ substitution handling.
239                 var substitutedDecision = substitutedHandler(matchedLink,
240                 ↪ newValue);

```

```

237         //         if (Equals(substitutedDecision, Constants.Break))
238         //             return Constants.Break;
239         //         if (Equals(substitutedDecision, Constants.Continue))
240         //         {
241         //             // Actual update here
242         //             Memory.SetLinkValue(newValue);
243         //         }
244         //         if (Equals(substitutedDecision, Constants.Skip))
245         //         {
246         //             // Cancel the update. TODO: decide use separate Cancel
247         //             ↪ constant or Skip is enough?
248         //         }
249         //     }
250     }
251 }
252 return _constants.Continue;
253 }
254
255 /// <summary>
256 /// <para>
257 /// Triggers the pattern or condition.
258 /// </para>
259 /// <para></para>
260 /// </summary>
261 /// <param name="patternOrCondition">
262 /// <para>The pattern or condition.</para>
263 /// <para></para>
264 /// </param>
265 /// <param name="matchHandler">
266 /// <para>The match handler.</para>
267 /// <para></para>
268 /// </param>
269 /// <param name="substitution">
270 /// <para>The substitution.</para>
271 /// <para></para>
272 /// </param>
273 /// <param name="substitutionHandler">
274 /// <para>The substitution handler.</para>
275 /// <para></para>
276 /// </param>
277 /// <exception cref="NotImplementedException">
278 /// <para></para>
279 /// <para></para>
280 /// </exception>
281 /// <exception cref="NotSupportedException">
282 /// <para></para>
283 /// <para></para>
284 /// </exception>
285 /// <exception cref="NotSupportedException">
286 /// <para></para>
287 /// <para></para>
288 /// </exception>
289 /// <exception cref="NotSupportedException">
290 /// <para></para>
291 /// <para></para>
292 /// </exception>
293 /// <exception cref="NotSupportedException">
294 /// <para></para>
295 /// <para></para>
296 /// </exception>
297 /// <returns>
298 /// <para>The link</para>
299 /// <para></para>
300 /// </returns>
301 public TLink Trigger(IList<TLink> patternOrCondition, ReadHandler<TLink>? matchHandler,
302     ↪ IList<TLink> substitution, WriteHandler<TLink> substitutionHandler)
303 {
304     var constants = _constants;
305     if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
306     {
307         return constants.Continue;
308     }
309     else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
310     ↪ Check if it is a correct condition
311     {
312         // Or it only applies to trigger without matchHandler.
313         throw new NotImplementedException();

```

```

312 }
313 else if (!substitution.IsNullOrEmpty()) // Creation
314 {
315     var before = Array.Empty<TLink>();
316     // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
317     ↪ (пройти мимо) или пустить (взять)?
318     if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
319     ↪ constants.Break))
320     {
321         return constants.Break;
322     }
323     var after = (IList<TLink>)substitution.ToArray();
324     if (_equalityComparer.Equals(after[0], default))
325     {
326         var newLink = _links.Create();
327         after[0] = newLink;
328     }
329     if (substitution.Count == 1)
330     {
331         after = _links.GetLink(substitution[0]);
332     }
333     else if (substitution.Count == 3)
334     {
335         //Links.Create(after);
336     }
337     else
338     {
339         throw new NotSupportedException();
340     }
341     return matchHandler != null ? substitutionHandler(before, after) :
342     ↪ constants.Continue;
343 }
344 else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
345 {
346     if (patternOrCondition.Count == 1)
347     {
348         var linkToDelete = patternOrCondition[0];
349         var before = _links.GetLink(linkToDelete);
350         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
351         ↪ constants.Break))
352         {
353             return constants.Break;
354         }
355         var after = Array.Empty<TLink>();
356         _links.Update(linkToDelete, constants.Null, constants.Null);
357         _links.Delete(linkToDelete);
358         return matchHandler != null ? substitutionHandler(before, after) :
359         ↪ constants.Continue;
360     }
361     else
362     {
363         throw new NotSupportedException();
364     }
365 }
366 else // Replace / Update
367 {
368     if (patternOrCondition.Count == 1) //-V3125
369     {
370         var linkToUpdate = patternOrCondition[0];
371         var before = _links.GetLink(linkToUpdate);
372         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
373         ↪ constants.Break))
374         {
375             return constants.Break;
376         }
377         var after = (IList<TLink>)substitution.ToArray(); //-V3125
378         if (_equalityComparer.Equals(after[0], default))
379         {
380             after[0] = linkToUpdate;
381         }
382         if (substitution.Count == 1)
383         {
384             if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
385             {
386                 after = _links.GetLink(substitution[0]);
387                 _links.Update(linkToUpdate, constants.Null, constants.Null);
388                 _links.Delete(linkToUpdate);
389             }
390         }
391     }
392 }

```



```

384     }
385     else if (substitution.Count == 3)
386     {
387         //Links.Update(after);
388     }
389     else
390     {
391         throw new NotSupportedException();
392     }
393     return matchHandler != null ? substitutionHandler(before, after) :
        ↪ constants.Continue;
394 }
395 else
396 {
397     throw new NotSupportedException();
398 }
399 }
400 }
401
402 /// <remarks>
403 /// IList[IList[IList[T]]]
404 /// | | | | |
405 /// | | | | |
406 /// | | | | |
407 /// | | | | |
408 /// | | | | |
409 /// | | | | |
410 /// | | | | |
411 /// | | | | |
412 public IList<IList<IList<TLink>>> Trigger(IList<TLink> condition, IList<TLink>
    ↪ substitution)
413 {
414     var changes = new List<IList<IList<TLink>>>();
415     var @continue = _constants.Continue;
416     Trigger(condition, AlwaysContinue, substitution, (before, after) =>
417     {
418         var change = new[] { before, after };
419         changes.Add(change);
420         return @continue;
421     });
422     return changes;
423 }
424 private TLink AlwaysContinue(IList<TLink> linkToMatch) => _constants.Continue;
425 }
426 }

```

1.18 ./csharp/Platform.Data.Doublets/Doublet.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.Doublets
8 {
9
10     /// <summary>
11     /// <para>.</para>
12     /// <para>.</para>
13     /// </summary>
14     /// <typeparam>
15     /// <para>.</para>
16     /// <para>.</para>
17     /// </typeparam>
18     public struct Doublet<T> : IEquatable<Doublet<T>>
19     {
20         private static readonly EqualityComparer<T> _equalityComparer =
            ↪ EqualityComparer<T>.Default;
21
22         /// <summary>
23         /// <para>.</para>
24         /// <para>.</para>
25         /// </summary>
26         /// <typeparam name="T">
27         /// <para>.</para>
28         /// <para>.</para>
29         /// </typeparam>
30         public readonly T Source;
31

```

```

32    /// <summury>
33    /// <para>.</para>
34    /// <para>.</para>
35    /// </summury>
36    /// <typeparam name="T">
37    /// <para>.</para>
38    /// <para>.</para>
39    /// </typeparam>
40    public readonly T Target;
41
42    /// <summury>
43    /// <para>.</para>
44    /// <para>.</para>
45    /// </summury>
46    /// <typeparam name="T">
47    /// <para>.</para>
48    /// <para>.</para>
49    /// </typeparam>
50    /// <param name="source">
51    /// <para>.</para>
52    /// <para>.</para>
53    /// </param>
54    /// <param name="target">
55    /// <para>.</para>
56    /// <para>.</para>
57    /// </param>
58    [MethodImpl(MethodImplOptions.AggressiveInlining)]
59    public Doublet(T source, T target)
60    {
61        Source = source;
62        Target = target;
63    }
64
65    /// <summury>
66    /// <para>.</para>
67    /// <para>.</para>
68    /// </summury>
69    /// <returns>
70    /// <para>.</para>
71    /// <para>.</para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    public override string ToString() => $"{Source}->{Target}";
75
76    /// <summury>
77    /// <para>.</para>
78    /// <para>.</para>
79    /// </summury>
80    /// <typeparam>
81    /// <para>.</para>
82    /// <para>.</para>
83    /// </typeparam>
84    /// <param name="other">
85    /// <para>.</para>
86    /// <para>.</para>
87    /// </param>
88    /// <returns>
89    /// <para>.</para>
90    /// <para>.</para>
91    /// </returns>
92    [MethodImpl(MethodImplOptions.AggressiveInlining)]
93    public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
94    ↪ && _equalityComparer.Equals(Target, other.Target);
95
96    /// <summury>
97    /// <para>.</para>
98    /// <para>.</para>
99    /// </summury>
100    /// <typeparam>
101    /// <para>.</para>
102    /// <para>.</para>
103    /// </typeparam>
104    /// <param name="obj">
105    /// <para>.</para>
106    /// <para>.</para>
107    /// </param>
108    /// <returns>
109    /// <para>.</para>

```

```

109     /// <para>.</para>
110     /// </returns>
111     [MethodImpl(MethodImplOptions.AggressiveInlining)]
112     public override bool Equals(object obj) => obj is Doublet<T> doublet ?
        ↳ base.Equals(doublet) : false;
113
114     /// <summary>
115     /// <para>.</para>
116     /// <para>.</para>
117     /// </summary>
118     /// <returns>
119     /// <para>.</para>
120     /// <para>.</para>
121     /// </returns>
122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
123     public override int GetHashCode() => (Source, Target).GetHashCode();
124
125     /// <summary>
126     /// <para>.</para>
127     /// <para>.</para>
128     /// </summary>
129     /// <param name="left">
130     /// <para>.</para>
131     /// <para>.</para>
132     /// </param>
133     /// <param name="right">
134     /// <para>.</para>
135     /// <para>.</para>
136     /// </param>
137     /// <returns>
138     /// <para>.</para>
139     /// <para>.</para>
140     /// </returns>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);
143
144     /// <summary>
145     /// <para>.</para>
146     /// <para>.</para>
147     /// </summary>
148     /// <param name="left">
149     /// <para>.</para>
150     /// <para>.</para>
151     /// </param>
152     /// <param name="right">
153     /// <para>.</para>
154     /// <para>.</para>
155     /// </param>
156     /// <returns>
157     /// <para>.</para>
158     /// <para>.</para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
162 }
163 }

```

1.19 ./csharp/Platform.Data.Doublets/DoubletComparer.cs

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets
7 {
8     /// <remarks>
9     /// TODO: Может стоит попробовать ref во всех методах (IRefEqualityComparer)
10    /// 2x faster with comparer
11    /// </remarks>
12    public class DoubletComparer<T> : IEqualityComparer<Doublet<T>>
13    {
14        /// <summary>
15        /// <para>
16        /// The .
17        /// </para>
18        /// <para></para>
19        /// </summary>
20        public static readonly DoubletComparer<T> Default = new DoubletComparer<T>();

```

```

21
22     /// <summary>
23     /// <para>
24     /// Determines whether this instance equals.
25     /// </para>
26     /// <para></para>
27     /// </summary>
28     /// <param name="x">
29     /// <para>The .</para>
30     /// <para></para>
31     /// </param>
32     /// <param name="y">
33     /// <para>The .</para>
34     /// <para></para>
35     /// </param>
36     /// <returns>
37     /// <para>The bool</para>
38     /// <para></para>
39     /// </returns>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public bool Equals(Doublet<T> x, Doublet<T> y) => x.Equals(y);
42
43     /// <summary>
44     /// <para>
45     /// Gets the hash code using the specified obj.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="obj">
50     /// <para>The obj.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The int</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public int GetHashCode(Doublet<T> obj) => obj.GetHashCode();
59 }
60 }

```

1.20 ./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.InteropServices;
5  using Platform.Converters;
6  using Platform.Delegates;
7  using Platform.Disposables;
8
9  namespace Platform.Data.Doublets.FFI
10 {
11     using TLink = System.UInt32;
12
13     public class UInt32UnitedMemoryLinks : DisposableBase, ILinks<TLink>
14     {
15         public LinksConstants<TLink> Constants { get; }
16
17         private readonly unsafe void* _ptr;
18
19         public UInt32UnitedMemoryLinks(string path)
20         {
21             unsafe
22             {
23                 _ptr = Methods.UInt32UnitedMemoryLinks_New(path);
24
25                 // TODO: Update api
26                 Constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
27             }
28         }
29
30         public TLink Count(ICollection<TLink>? restriction)
31         {
32             unsafe
33             {
34                 var array = stackalloc uint[restriction.Count];
35                 for (var i = 0; i < restriction.Count; i++)
36                 {
37                     array[i] = restriction[i];
38                 }
39             }
40         }
41     }
42 }

```

```

38     }
39     return Methods.UInt32UnitedMemoryLinks_Count(_ptr, array,
    ↪ (nuint)(restriction?.Count ?? 0));
40 }
41 }
42
43 public TLink Each(IList<TLink> restriction, ReadHandler<TLink>? handler)
44 {
45     unsafe
46     {
47         Methods.EachCallback_UInt32 callback = (link) => handler?.Invoke(new
    ↪ Link<TLink>(link.Index, link.Source, link.Target)) ?? Constants.Continue;
48         var array = stackalloc uint[restriction.Count];
49         for (var i = 0; i < restriction.Count; i++)
50         {
51             array[i] = restriction[i];
52         }
53         return Methods.UInt32UnitedMemoryLinks_Each(_ptr, array,
    ↪ (nuint)(restriction?.Count ?? 0), callback);
54     }
55 }
56
57 public TLink Create(IList<TLink> substitution, WriteHandler<TLink> handler)
58 {
59     unsafe
60     {
61         Methods.CreateCallback_UInt32 callback = (before, after) => handler != null ?
    ↪ handler(new Link<TLink>(before.Index, before.Source, before.Target), new
    ↪ Link<TLink>(after.Index, after.Source, after.Target)) : Constants.Continue;
62         fixed (uint* substitutionPtr = (uint[])substitution)
63         {
64             return Methods.UInt32UnitedMemoryLinks_Create(_ptr, substitutionPtr,
    ↪ (nuint)(substitution?.Count ?? 0), callback);
65         }
66     }
67 }
68
69 public TLink Update(IList<TLink> restriction, IList<TLink> substitution,
    ↪ WriteHandler<TLink> handler)
70 {
71     unsafe
72     {
73         var restrictionArray = stackalloc uint[restriction.Count];
74         for (var i = 0; i < restriction.Count; i++)
75         {
76             restrictionArray[i] = restriction[i];
77         }
78         var substitutionArray = stackalloc uint[substitution.Count];
79         for (var i = 0; i < restriction.Count; i++)
80         {
81             substitutionArray[i] = restriction[i];
82         }
83         Methods.UpdateCallback_UInt32 callback = (before, after) => handler?.Invoke(new
    ↪ Link<TLink>(before.Index, before.Source, before.Target), new
    ↪ Link<TLink>(after.Index, after.Source, after.Target)) ?? Constants.Continue;
84         return Methods.UInt32UnitedMemoryLinks_Update(_ptr, restrictionArray,
    ↪ (nuint)(restriction?.Count ?? 0), substitutionArray,
    ↪ (nuint)(substitution?.Count ?? 0), callback);
85     }
86 }
87
88 public TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler)
89 {
90     unsafe
91     {
92         var restrictionArray = stackalloc uint[restriction.Count];
93         for (var i = 0; i < restriction.Count; i++)
94         {
95             restrictionArray[i] = restriction[i];
96         }
97         Methods.DeleteCallback_UInt32 callback = (before, after) => handler?.Invoke(new
    ↪ Link<TLink>(before.Index, before.Source, before.Target), new
    ↪ Link<TLink>(after.Index, after.Source, after.Target)) ?? Constants.Continue;
98         return Methods.UInt32UnitedMemoryLinks_Delete(_ptr, restrictionArray,
    ↪ (nuint)(restriction?.Count ?? 0), callback);
99     }
100 }

```

```

101     protected override void Dispose(bool manual, bool wasDisposed)
102     {
103         unsafe
104         {
105             if (wasDisposed || _ptr == null)
106             {
107                 return;
108             }
109             Methods.UInt32UnitedMemoryLinks_Drop(_ptr);
110         }
111     }
112 }
113 }
114 }

```

1.21 ./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.InteropServices;
5  using Platform.Converters;
6  using Platform.Delegates;
7  using Platform.Disposables;
8
9  namespace Platform.Data.Doublets.FFI
10 {
11     struct FfiLink_UInt8
12     {
13         public Byte Index;
14         public Byte Source;
15         public Byte Target;
16     }
17
18     struct FfiLink_UInt16
19     {
20         public UInt16 Index;
21         public UInt16 Source;
22         public UInt16 Target;
23     }
24
25     struct FfiLink_UInt32
26     {
27         public UInt32 Index;
28         public UInt32 Source;
29         public UInt32 Target;
30     }
31
32     struct FfiLink_UInt64
33     {
34         public UInt64 Index;
35         public UInt64 Source;
36         public UInt64 Target;
37     }
38
39     unsafe static class Methods
40     {
41         private const string DllName = "Platform.Doublets";
42
43         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
44         public delegate Byte EachCallback_UInt8(FfiLink_UInt8 link);
45
46         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
47         public delegate UInt16 EachCallback_UInt16(FfiLink_UInt16 link);
48
49         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
50         public delegate UInt32 EachCallback_UInt32(FfiLink_UInt32 link);
51
52         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
53         public delegate UInt64 EachCallback_UInt64(FfiLink_UInt64 link);
54
55         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
56         public delegate Byte CreateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
57
58         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
59         public delegate UInt16 CreateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
60             ↪ after);
61
62         [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
63         public delegate UInt32 CreateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
64             ↪ after);
65
66     }
67 }

```

```

64 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
65 public delegate UInt64 CreateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
    ↪ after);
66
67 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
68 public delegate Byte UpdateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
69
70 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
71 public delegate UInt16 UpdateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
    ↪ after);
72
73 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
74 public delegate UInt32 UpdateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
    ↪ after);
75
76 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
77 public delegate UInt64 UpdateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
    ↪ after);
78
79 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
80 public delegate Byte DeleteCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
81
82 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
83 public delegate UInt16 DeleteCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
    ↪ after);
84
85 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
86 public delegate UInt32 DeleteCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
    ↪ after);
87
88 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
89 public delegate UInt64 DeleteCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
    ↪ after);
90
91 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
92 public static extern void* ByteUnitedMemoryLinks_New(string path);
93
94 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
95 public static extern void* UInt16UnitedMemoryLinks_New(string path);
96
97 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
98 public static extern void* UInt32UnitedMemoryLinks_New(string path);
99
100 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
101 public static extern void* UInt64UnitedMemoryLinks_New(string path);
102
103 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
104 public static extern void ByteUnitedMemoryLinks_Drop(void* self);
105
106 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
107 public static extern void UInt16UnitedMemoryLinks_Drop(void* self);
108
109 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
110 public static extern void UInt32UnitedMemoryLinks_Drop(void* self);
111
112 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
113 public static extern void UInt64UnitedMemoryLinks_Drop(void* self);
114
115 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
116 public static extern byte ByteUnitedMemoryLinks_Create(void* self, byte* substitution,
    ↪ nuint substitutionLength, CreateCallback_UInt8 callback);
117
118 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
119 public static extern ushort UInt16UnitedMemoryLinks_Create(void* self, ushort*
    ↪ substitution, nuint substitutionLength, CreateCallback_UInt16 callback);
120
121 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
122 public static extern uint UInt32UnitedMemoryLinks_Create(void* self, uint* substitution,
    ↪ nuint substitutionLength, CreateCallback_UInt32 callback);
123
124 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
125 public static extern ulong UInt64UnitedMemoryLinks_Create(void* self, ulong*
    ↪ substitution, nuint substitutionLength, CreateCallback_UInt64 callback);
126
127 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
128 public static extern byte ByteUnitedMemoryLinks_Count(void* self, byte* restriction,
    ↪ nuint len);
129

```

```

130 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
131 public static extern ushort UInt16UnitedMemoryLinks_Count(void* self, ushort*
    ↳ restriction, nuint len);
132
133 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
134 public static extern uint UInt32UnitedMemoryLinks_Count(void* self, uint* restriction,
    ↳ nuint len);
135
136 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
137 public static extern ulong UInt64UnitedMemoryLinks_Count(void* self, ulong* restriction,
    ↳ nuint len);
138
139 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
140 public static extern byte ByteUnitedMemoryLinks_Each(void* self, byte* restriction,
    ↳ nuint len, EachCallback_UInt8 callback);
141
142 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
143 public static extern ushort UInt16UnitedMemoryLinks_Each(void* self, ushort*
    ↳ restriction, nuint len, EachCallback_UInt16 callback);
144
145 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
146 public static extern uint UInt32UnitedMemoryLinks_Each(void* self, uint* restriction,
    ↳ nuint len, EachCallback_UInt32 callback);
147
148 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
149 public static extern ulong UInt64UnitedMemoryLinks_Each(void* self, ulong* restriction,
    ↳ nuint len, EachCallback_UInt64 callback);
150
151 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
152 public static extern byte ByteUnitedMemoryLinks_Update(void* self, byte* restriction,
    ↳ nuint restrictionLength, byte* substitution, nuint substitutionLength,
    ↳ UpdateCallback_UInt8 callback);
153
154 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
155 public static extern ushort UInt16UnitedMemoryLinks_Update(void* self, ushort*
    ↳ restriction, nuint restrictionLength, ushort* substitution, nuint
    ↳ substitutionLength, UpdateCallback_UInt16 callback);
156
157 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
158 public static extern uint UInt32UnitedMemoryLinks_Update(void* self, uint* restriction,
    ↳ nuint restrictionLength, uint* substitution, nuint substitutionLength,
    ↳ UpdateCallback_UInt32 callback);
159
160 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
161 public static extern ulong UInt64UnitedMemoryLinks_Update(void* self, ulong*
    ↳ restriction, nuint restrictionLength, ulong* substitution, nuint
    ↳ substitutionLength, UpdateCallback_UInt64 callback);
162
163 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
164 public static extern byte ByteUnitedMemoryLinks_Delete(void* self, byte* restriction,
    ↳ nuint restrictionLength, DeleteCallback_UInt8 callback);
165
166 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
167 public static extern ushort UInt16UnitedMemoryLinks_Delete(void* self, ushort*
    ↳ restriction, nuint len, DeleteCallback_UInt16 callback);
168
169 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
170 public static extern uint UInt32UnitedMemoryLinks_Delete(void* self, uint* restriction,
    ↳ nuint len, DeleteCallback_UInt32 callback);
171
172 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
173 public static extern ulong UInt64UnitedMemoryLinks_Delete(void* self, ulong*
    ↳ restriction, nuint len, DeleteCallback_UInt64 callback);
174 }

```

```

175
176 public class UnitedMemoryLinks<TLink> : DisposableBase, ILinks<TLink>
177 {
178     private static readonly UncheckedConverter<byte, TLink> from_u8 =
        ↳ UncheckedConverter<byte, TLink>.Default;
179     private static readonly UncheckedConverter<ushort, TLink> from_u16 =
        ↳ UncheckedConverter<ushort, TLink>.Default;
180     private static readonly UncheckedConverter<uint, TLink> from_u32 =
        ↳ UncheckedConverter<uint, TLink>.Default;
181     private static readonly UncheckedConverter<ulong, TLink> from_u64 =
        ↳ UncheckedConverter<ulong, TLink>.Default;
182     private static readonly UncheckedConverter<TLink, ulong> from_t =
        ↳ UncheckedConverter<TLink, ulong>.Default;
183

```



```

184 public LinksConstants<TLink> Constants { get; }
185
186 private readonly unsafe void* _ptr;
187
188 public UnitedMemoryLinks(string path)
189 {
190     TLink t = default;
191     unsafe
192     {
193         _ptr = t switch
194         {
195             byte => Methods.ByteUnitedMemoryLinks_New(path),
196             ushort => Methods.UInt16UnitedMemoryLinks_New(path),
197             uint => Methods.UInt32UnitedMemoryLinks_New(path),
198             ulong => Methods.UInt64UnitedMemoryLinks_New(path),
199             _ => throw new NotImplementedException()
200         };
201
202         // TODO: Update api
203         Constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
204     }
205 }
206
207 public TLink Count(IList<TLink>? restriction)
208 {
209     var restrictionLength = restriction?.Count ?? 0;
210     unsafe
211     {
212         TLink t = default;
213         switch (t)
214         {
215             case byte:
216             {
217                 var restrictionArray = stackalloc byte[restrictionLength];
218                 var byteRestrictionArray = (IList<byte>)restriction;
219                 for (var i = 0; i < restrictionLength; i++)
220                 {
221                     restrictionArray[i] = byteRestrictionArray[i];
222                 }
223                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Count(_ptr,
224                                     ↪ restrictionArray, (nuint)restrictionLength));
225             }
226             case ushort:
227             {
228                 var restrictionArray = stackalloc ushort[restrictionLength];
229                 var ushortRestrictionArray = (IList<ushort>)restriction;
230                 for (var i = 0; i < restrictionLength; i++)
231                 {
232                     restrictionArray[i] = ushortRestrictionArray[i];
233                 }
234                 return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Count(_ptr,
235                                     ↪ restrictionArray, (nuint)restrictionLength));
236             }
237             case uint:
238             {
239                 var restrictionArray = stackalloc uint[restrictionLength];
240                 var uintRestrictionArray = (IList<uint>)restriction;
241                 for (var i = 0; i < restrictionLength; i++)
242                 {
243                     restrictionArray[i] = uintRestrictionArray[i];
244                 }
245                 return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Count(_ptr,
246                                     ↪ restrictionArray, (nuint)restrictionLength));
247             }
248             case ulong:
249             {
250                 {
251                     var restrictionArray = stackalloc ulong[restrictionLength];
252                     var ulongRestrictionArray = (IList<ulong>)restriction;
253                     for (var i = 0; i < restrictionLength; i++)
254                     {
255                         restrictionArray[i] = ulongRestrictionArray[i];
256                     }
257                     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Count(_ptr,
258                                     ↪ restrictionArray, (nuint)restrictionLength));
259                 }
260             }
261             default:
262             {

```

```

259         throw new NotImplementedException();
260     }
261 }
262 }
263 }
264
265 public TLink Each(IList<TLink> restriction, ReadHandler<TLink>? handler)
266 {
267     var restrictionLength = restriction?.Count ?? 0;
268     unsafe
269     {
270         TLink t = default;
271         switch (t)
272         {
273             case byte:
274             {
275                 byte Callback(FfiLink_UInt8 link) => (byte)from_t.Convert(handler !=
                ↪ null ? handler(new Link<TLink>(from_u8.Convert(link.Index),
                ↪ from_u8.Convert(link.Source), from_u8.Convert(link.Target))) :
                ↪ Constants.Continue);
276                 var restrictionArray = stackalloc byte[restrictionLength];
277                 var byteRestrictionArray = (IList<byte>)restriction;
278                 for (var i = 0; i < restrictionLength; i++)
279                 {
280                     restrictionArray[i] = byteRestrictionArray[i];
281                 }
282                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Each(_ptr,
                ↪ restrictionArray, (nuint)restrictionLength, Callback));
283             }
284             case ushort:
285             {
286                 ushort Callback(FfiLink_UInt16 link) => (ushort)from_t.Convert(handler
                ↪ != null ? handler(new Link<TLink>(from_u16.Convert(link.Index),
                ↪ from_u16.Convert(link.Source), from_u16.Convert(link.Target))) :
                ↪ Constants.Continue);
287                 var restrictionArray = stackalloc ushort[restrictionLength];
288                 var ushortRestrictionArray = (IList<ushort>)restriction;
289                 for (var i = 0; i < restrictionLength; i++)
290                 {
291                     restrictionArray[i] = ushortRestrictionArray[i];
292                 }
293                 return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Each(_ptr,
                ↪ restrictionArray, (nuint)restrictionLength, Callback));
294             }
295             case uint:
296             {
297                 uint Callback(FfiLink_UInt32 link) => (uint)from_t.Convert(handler !=
                ↪ null ? handler(new Link<TLink>(from_u32.Convert(link.Index),
                ↪ from_u32.Convert(link.Source), from_u32.Convert(link.Target))) :
                ↪ Constants.Continue);
298                 var restrictionArray = stackalloc uint[restrictionLength];
299                 var uintRestrictionArray = (IList<uint>)restriction;
300                 for (var i = 0; i < restrictionLength; i++)
301                 {
302                     restrictionArray[i] = uintRestrictionArray[i];
303                 }
304                 return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Each(_ptr,
                ↪ restrictionArray, (nuint)restrictionLength, Callback));
305             }
306             case ulong:
307             {
308                 {
309                     ulong Callback(FfiLink_UInt64 link) => from_t.Convert(handler !=
                ↪ null ? handler(new Link<TLink>(from_u64.Convert(link.Index),
                ↪ from_u64.Convert(link.Source), from_u64.Convert(link.Target))) :
                ↪ Constants.Continue);
310                     var restrictionArray = stackalloc UInt64[restrictionLength];
311                     var ulongRestrictionArray = (IList<ulong>)restriction;
312                     for (var i = 0; i < restrictionLength; i++)
313                     {
314                         restrictionArray[i] = ulongRestrictionArray[i];
315                     }
316                     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Each(_ptr,
                ↪ restrictionArray, (nuint)restrictionLength, Callback));
317                 }
318             }
319             default:
320             {

```

```

321         throw new NotImplementedException();
322     }
323 }
324 }
325 }
326
327 public TLink Create(IList<TLink> substitution, WriteHandler<TLink> handler)
328 {
329     var substitutionLength = substitution?.Count ?? 0;
330     unsafe
331     {
332         TLink t = default;
333         switch (t)
334         {
335             case byte:
336             {
337                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
338                     (byte)from_t.Convert(handler != null ? handler(new
339                     ↪ Link<TLink>(from_u8.Convert(before.Index),
340                     ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
341                     ↪ Link<TLink>(from_u8.Convert(after.Index),
342                     ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
343                     ↪ Constants.Continue);
344                 var substitutionArray = stackalloc byte[substitutionLength];
345                 var byteSubstitutionArray = (IList<byte>)substitution;
346                 for (var i = 0; i < substitutionLength; i++)
347                 {
348                     substitutionArray[i] = byteSubstitutionArray[i];
349                 }
350                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Create(_ptr,
351                     ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
352             }
353             case ushort:
354             {
355                 ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
356                     (ushort)from_t.Convert(handler != null ? handler(new
357                     ↪ Link<TLink>(from_u16.Convert(before.Index),
358                     ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
359                     ↪ new Link<TLink>(from_u16.Convert(after.Index),
360                     ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
361                     ↪ Constants.Continue);
362                 var substitutionArray = stackalloc ushort[substitutionLength];
363                 var ushortSubstitutionArray = (IList<ushort>)substitution;
364                 for (var i = 0; i < substitutionLength; i++)
365                 {
366                     substitutionArray[i] = ushortSubstitutionArray[i];
367                 }
368                 return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Create(_ptr,
369                     ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
370             }
371             case uint:
372             {
373                 uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
374                     (uint)from_t.Convert(handler != null ? handler(new
375                     ↪ Link<TLink>(from_u32.Convert(before.Index),
376                     ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
377                     ↪ new Link<TLink>(from_u32.Convert(after.Index),
378                     ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
379                     ↪ Constants.Continue);
380                 var substitutionArray = stackalloc uint[substitutionLength];
381                 var uintSubstitutionArray = (IList<uint>)substitution;
382                 for (var i = 0; i < substitutionLength; i++)
383                 {
384                     substitutionArray[i] = uintSubstitutionArray[i];
385                 }
386                 return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Create(_ptr,
387                     ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
388             }
389             case ulong:
390             {
391                 ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
392                     (ulong)from_t.Convert(handler != null ? handler(new
393                     ↪ Link<TLink>(from_u64.Convert(before.Index),
394                     ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
395                     ↪ new Link<TLink>(from_u64.Convert(after.Index),
396                     ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
397                     ↪ Constants.Continue);

```

```

372     var substitutionArray = stackalloc ulong[substitutionLength];
373     var ulongSubstitutionArray = (IList<ulong>)substitution;
374     for (var i = 0; i < substitutionLength; i++)
375     {
376         substitutionArray[i] = ulongSubstitutionArray[i];
377     }
378     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
379         ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
380 }
381 default:
382 {
383     throw new NotImplementedException();
384 }
385 };
386 }
387 }
388
389 public TLink Update(IList<TLink> restriction, IList<TLink> substitution,
390     ↳ WriteHandler<TLink> handler)
391 {
392     var restrictionLength = restriction?.Count ?? 0;
393     var substitutionLength = substitution?.Count ?? 0;
394     unsafe
395     {
396         TLink t = default;
397         switch (t)
398         {
399             case byte:
400             {
401                 var restrictionArray = stackalloc byte[restrictionLength];
402                 var byteRestrictionArray = (IList<byte>)restriction;
403                 for (var i = 0; i < restrictionLength; i++)
404                 {
405                     restrictionArray[i] = byteRestrictionArray[i];
406                 }
407                 var substitutionArray = stackalloc byte[substitutionLength];
408                 var byteSubstitutionArray = (IList<byte>)substitution;
409                 for (var i = 0; i < substitutionLength; i++)
410                 {
411                     substitutionArray[i] = byteSubstitutionArray[i];
412                 }
413                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
414                     ↳ (byte)from_t.Convert(handler != null ? handler(new
415                     ↳ Link<TLink>(from_u8.Convert(before.Index),
416                     ↳ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
417                     ↳ Link<TLink>(from_u8.Convert(after.Index),
418                     ↳ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
419                     ↳ Constants.Continue);
420                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Update(_ptr,
421                     ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
422                     ↳ (nuint)(substitution?.Count ?? 0), Callback));
423             }
424             case ushort:
425             {
426                 var restrictionArray = stackalloc ushort[restrictionLength];
427                 var ushortRestrictionArray = (IList<ushort>)restriction;
428                 for (var i = 0; i < restrictionLength; i++)
429                 {
430                     restrictionArray[i] = ushortRestrictionArray[i];
431                 }
432                 var substitutionArray = stackalloc ushort[substitutionLength];
433                 var ushortSubstitutionArray = (IList<ushort>)substitution;
434                 for (var i = 0; i < substitutionLength; i++)
435                 {
436                     substitutionArray[i] = ushortSubstitutionArray[i];
437                 }
438                 ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
439                     ↳ (ushort)from_t.Convert(handler != null ? handler(new
440                     ↳ Link<TLink>(from_u16.Convert(before.Index),
441                     ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
442                     ↳ new Link<TLink>(from_u16.Convert(after.Index),
443                     ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
444                     ↳ Constants.Continue);
445                 return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Update(_ptr,
446                     ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
447                     ↳ (nuint)(substitution?.Count ?? 0), Callback));
448             }
449         }
450     }
451 }

```

```

432     case uint:
433     {
434         var restrictionArray = stackalloc uint[restrictionLength];
435         var uintRestrictionArray = (IList<uint>)restriction;
436         for (var i = 0; i < restrictionLength; i++)
437         {
438             restrictionArray[i] = uintRestrictionArray[i];
439         }
440         var substitutionArray = stackalloc uint[substitutionLength];
441         var uintSubstitutionArray = (IList<uint>)substitution;
442         for (var i = 0; i < substitutionLength; i++)
443         {
444             substitutionArray[i] = uintSubstitutionArray[i];
445         }
446         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
447         {
448             (uint)from_t.Convert(handler != null ? handler(new
449             ↪ Link<TLink>(from_u32.Convert(before.Index),
450             ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
451             ↪ new Link<TLink>(from_u32.Convert(after.Index),
452             ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
453             ↪ Constants.Continue);
454         return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Update(_ptr,
455             ↪ restrictionArray, (nuint)restrictionLength, substitutionArray,
456             ↪ (nuint)(substitution?.Count ?? 0), Callback));
457     }
458     case ulong:
459     {
460         var restrictionArray = stackalloc ulong[restrictionLength];
461         var ulongRestrictionArray = (IList<ulong>)restriction;
462         for (var i = 0; i < restrictionLength; i++)
463         {
464             restrictionArray[i] = ulongRestrictionArray[i];
465         }
466         var substitutionArray = stackalloc ulong[substitutionLength];
467         var ulongSubstitutionArray = (IList<ulong>)substitution;
468         for (var i = 0; i < substitutionLength; i++)
469         {
470             substitutionArray[i] = ulongSubstitutionArray[i];
471         }
472         ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
473         {
474             (ulong)from_t.Convert(handler != null ? handler(new
475             ↪ Link<TLink>(from_u64.Convert(before.Index),
476             ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
477             ↪ new Link<TLink>(from_u64.Convert(after.Index),
478             ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
479             ↪ Constants.Continue);
480         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Update(_ptr,
481             ↪ restrictionArray, (nuint)restrictionLength, substitutionArray,
482             ↪ (nuint)(substitution?.Count ?? 0), Callback));
483     }
484     default:
485     {
486         throw new NotImplementedException();
487     }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

490     byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
491         ↪ (byte)from_t.Convert(handler != null ? handler(new
492         ↪ Link<TLink>(from_u8.Convert(before.Index),
493         ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
494         ↪ Link<TLink>(from_u8.Convert(after.Index),
495         ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
496         ↪ Constants.Continue);
497     return (TLink)(object)Methods.ByteUnitedMemoryLinks_Delete(_ptr,
498     ↪ restrictionArray, (nuint)restrictionLength, Callback);
499 }
500 case ushort:
501 {
502     var restrictionArray = stackalloc ushort[restrictionLength];
503     var ushortRestrictionArray = (IList<ushort>)restriction;
504     for (var i = 0; i < restrictionLength; i++)
505     {
506         restrictionArray[i] = ushortRestrictionArray[i];
507     }
508     ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
509         ↪ (ushort)from_t.Convert(handler != null ? handler(new
510         ↪ Link<TLink>(from_u16.Convert(before.Index),
511         ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
512         ↪ new Link<TLink>(from_u16.Convert(after.Index),
513         ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
514         ↪ Constants.Continue);
515     return (TLink)(object)Methods.UInt16UnitedMemoryLinks_Delete(_ptr,
516     ↪ restrictionArray, (nuint)restrictionLength, Callback);
517 }
518 case uint:
519 {
520     var restrictionArray = stackalloc uint[restrictionLength];
521     var uintRestrictionArray = (IList<uint>)restriction;
522     for (var i = 0; i < restrictionLength; i++)
523     {
524         restrictionArray[i] = uintRestrictionArray[i];
525     }
526     uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
527         ↪ (uint)from_t.Convert(handler != null ? handler(new
528         ↪ Link<TLink>(from_u32.Convert(before.Index),
529         ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
530         ↪ new Link<TLink>(from_u32.Convert(after.Index),
531         ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
532         ↪ Constants.Continue);
533     return (TLink)(object)Methods.UInt32UnitedMemoryLinks_Delete(_ptr,
534     ↪ restrictionArray, (nuint)restrictionLength, Callback);
535 }
536 case ulong:
537 {
538     var restrictionArray = stackalloc ulong[restrictionLength];
539     var ulongRestrictionArray = (IList<ulong>)restriction;
540     for (var i = 0; i < restrictionLength; i++)
541     {
542         restrictionArray[i] = ulongRestrictionArray[i];
543     }
544     ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
545         ↪ (ulong)from_t.Convert(handler != null ? handler(new
546         ↪ Link<TLink>(from_u64.Convert(before.Index),
547         ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
548         ↪ new Link<TLink>(from_u64.Convert(after.Index),
549         ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
550         ↪ Constants.Continue);
551     return (TLink)(object)Methods.UInt64UnitedMemoryLinks_Delete(_ptr,
552     ↪ restrictionArray, (nuint)restrictionLength, Callback);
553 }
554 default:
555 {
556     throw new NotImplementedException();
557 }
558 }
559 }
560 }
561
562 protected override void Dispose(bool manual, bool wasDisposed)
563 {
564     unsafe
565     {
566         if (wasDisposed && _ptr != null)
567         {
568

```

```

540         return;
541     }
542     TLink t = default;
543     switch (t)
544     {
545         case byte:
546             Methods.ByteUnitedMemoryLinks_Drop(_ptr);
547             break;
548         case ushort:
549             Methods.UInt16UnitedMemoryLinks_Drop(_ptr);
550             break;
551         case uint:
552             Methods.UInt32UnitedMemoryLinks_Drop(_ptr);
553             break;
554         case ulong:
555             Methods.UInt64UnitedMemoryLinks_Drop(_ptr);
556             break;
557         default:
558             throw new NotImplementedException();
559     }
560 }
561 }
562 }
563 }

```

1.22 ./csharp/Platform.Data.Doublets/ILinks.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  using System.Collections.Generic;
4
5  namespace Platform.Data.Doublets
6  {
7      /// <summary>
8      /// <para>
9      /// Defines the links.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="ILinks{TLink, LinksConstants{TLink}}"/>
14     public interface ILinks<TLink> : ILinks<TLink, LinksConstants<TLink>>
15     {
16     }
17 }

```

1.23 ./csharp/Platform.Data.Doublets/ILinksExtensions.cs

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Runtime.CompilerServices;
6  using Platform.Ranges;
7  using Platform.Collections.Lists;
8  using Platform.Random;
9  using Platform.Setters;
10 using Platform.Converters;
11 using Platform.Numbers;
12 using Platform.Data.Exceptions;
13 using Platform.Data.Doublets.Decorators;
14 using Platform.Delegates;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets
19 {
20     /// <summary>
21     /// <para>
22     /// Represents the links extensions.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     public static class ILinksExtensions
27     {
28         /// <summary>
29         /// <para>
30         /// Runs the random creations using the specified links.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <typeparam name="TLink">
35         /// <para>The link.</para>

```

```

36     /// <para></para>
37     /// </typeparam>
38     /// <param name="links">
39     /// <para>The links.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="amountOfCreations">
43     /// <para>The amount of creations.</para>
44     /// <para></para>
45     /// </param>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
↳ amountOfCreations)
48     {
49         var random = RandomHelpers.Default;
50         var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
51         var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
52         for (var i = 0UL; i < amountOfCreations; i++)
53         {
54             var linksAddressRange = new Range<ulong>(0,
↳ addressToUInt64Converter.Convert(links.Count()));
55             var source =
↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
56             var target =
↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
57             links.GetOrCreate(source, target);
58         }
59     }
60
61     /// <summary>
62     /// <para>
63     /// Runs the random searches using the specified links.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <typeparam name="TLink">
68     /// <para>The link.</para>
69     /// <para></para>
70     /// </typeparam>
71     /// <param name="links">
72     /// <para>The links.</para>
73     /// <para></para>
74     /// </param>
75     /// <param name="amountOfSearches">
76     /// <para>The amount of searches.</para>
77     /// <para></para>
78     /// </param>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public static void RunRandomSearches<TLink>(this ILinks<TLink> links, ulong
↳ amountOfSearches)
81     {
82         var random = RandomHelpers.Default;
83         var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
84         var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
85         for (var i = 0UL; i < amountOfSearches; i++)
86         {
87             var linksAddressRange = new Range<ulong>(0,
↳ addressToUInt64Converter.Convert(links.Count()));
88             var source =
↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
89             var target =
↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
90             links.SearchOrDefault(source, target);
91         }
92     }
93
94     /// <summary>
95     /// <para>
96     /// Runs the random deletions using the specified links.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <typeparam name="TLink">
101    /// <para>The link.</para>
102    /// <para></para>
103    /// </typeparam>
104    /// <param name="links">
105    /// <para>The links.</para>

```



```

106 /// <para></para>
107 /// </param>
108 /// <param name="amountOfDeletions">
109 /// <para>The amount of deletions.</para>
110 /// <para></para>
111 /// </param>
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
    ↳ amountOfDeletions)
114 {
115     var random = RandomHelpers.Default;
116     var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
117     var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
118     var linksCount = addressToUInt64Converter.Convert(links.Count());
119     var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
120     for (var i = 0UL; i < amountOfDeletions; i++)
121     {
122         linksCount = addressToUInt64Converter.Convert(links.Count());
123         if (linksCount <= min)
124         {
125             break;
126         }
127         var linksAddressRange = new Range<ulong>(min, linksCount);
128         var link =
            ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
129         links.Delete(link);
130     }
131 }
132
133 /// <summary>
134 /// <para>
135 /// Deletes the links.
136 /// </para>
137 /// <para></para>
138 /// </summary>
139 /// <typeparam name="TLink">
140 /// <para>The link.</para>
141 /// <para></para>
142 /// </typeparam>
143 /// <param name="links">
144 /// <para>The links.</para>
145 /// <para></para>
146 /// </param>
147 /// <param name="linkToDelete">
148 /// <para>The link to delete.</para>
149 /// <para></para>
150 /// </param>
151 [MethodImpl(MethodImplOptions.AggressiveInlining)]
152 public static TLink Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete,
    ↳ WriteHandler<TLink>? handler)
153 {
154     if (links.Exists(linkToDelete))
155     {
156         links.EnforceResetValues(linkToDelete, handler);
157     }
158     return links.Delete(new LinkAddress<TLink>(linkToDelete), handler);
159 }
160
161 /// <remarks>
162 /// TODO: Возможно есть очень простой способ это сделать.
163 /// (Например просто удалить файл, или изменить его размер таким образом,
164 /// чтобы удалился весь контент)
165 /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
166 /// </remarks>
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 public static void DeleteAll<TLink>(this ILinks<TLink> links)
169 {
170     var equalityComparer = EqualityComparer<TLink>.Default;
171     var comparer = Comparer<TLink>.Default;
172     for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
        ↳ Arithmetic.Decrement(i))
173     {
174         links.Delete(i);
175         if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
176         {
177             i = links.Count();
178         }
179     }

```

```

180 }
181
182 /// <summary>
183 /// <para>
184 /// Firsts the links.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 /// <typeparam name="TLink">
189 /// <para>The link.</para>
190 /// <para></para>
191 /// </typeparam>
192 /// <param name="links">
193 /// <para>The links.</para>
194 /// <para></para>
195 /// </param>
196 /// <exception cref="InvalidOperationException">
197 /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
198 /// <para></para>
199 /// </exception>
200 /// <exception cref="InvalidOperationException">
201 /// <para>В хранилище нет связей.</para>
202 /// <para></para>
203 /// </exception>
204 /// <returns>
205 /// <para>The first link.</para>
206 /// <para></para>
207 /// </returns>
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 public static TLink First<TLink>(this ILinks<TLink> links)
210 {
211     TLink firstLink = default;
212     var equalityComparer = EqualityComparer<TLink>.Default;
213     if (equalityComparer.Equals(links.Count(), default))
214     {
215         throw new InvalidOperationException("В хранилище нет связей.");
216     }
217     links.Each(links.Constants.Any, links.Constants.Any, link =>
218     {
219         firstLink = link[links.Constants.IndexPart];
220         return links.Constants.Break;
221     });
222     if (equalityComparer.Equals(firstLink, default))
223     {
224         throw new InvalidOperationException("В процессе поиска по хранилищу не было
225             ↳ найдено связей.");
226     }
227     return firstLink;
228 }
229
230 /// <summary>
231 /// <para>
232 /// Singles the or default using the specified links.
233 /// </para>
234 /// <para></para>
235 /// </summary>
236 /// <typeparam name="TLink">
237 /// <para>The link.</para>
238 /// <para></para>
239 /// </typeparam>
240 /// <param name="links">
241 /// <para>The links.</para>
242 /// <para></para>
243 /// </param>
244 /// <param name="query">
245 /// <para>The query.</para>
246 /// <para></para>
247 /// </param>
248 /// <returns>
249 /// <para>The result.</para>
250 /// <para></para>
251 /// </returns>
252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
253 public static IList<TLink> SingleOrDefault<TLink>(this ILinks<TLink> links, IList<TLink>
254     ↳ query)
255 {
256     IList<TLink> result = null;
257     var count = 0;

```

```

256     var constants = links.Constants;
257     var @continue = constants.Continue;
258     var @break = constants.Break;
259     links.Each(query, linkHandler);
260     return result;
261
262     TLink linkHandler(IList<TLink> link)
263     {
264         if (count == 0)
265         {
266             result = link;
267             count++;
268             return @continue;
269         }
270         else
271         {
272             result = null;
273             return @break;
274         }
275     }
276 }
277
278 #region Paths
279
280 /// <remarks>
281 /// TODO: Как так? Как то что ниже может быть корректно?
282 /// Скорее всего практически не применимо
283 /// Предполагалось, что можно было конвертировать формируемый в проходе через
284   ↳ SequenceWalker
285 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
286 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
287 /// </remarks>
288 [MethodImpl(MethodImplOptions.AggressiveInlining)]
289 public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
290   ↳ path)
291 {
292     var current = path[0];
293     //EnsureLinkExists(current, "path");
294     if (!links.Exists(current))
295     {
296         return false;
297     }
298     var equalityComparer = EqualityComparer<TLink>.Default;
299     var constants = links.Constants;
300     for (var i = 1; i < path.Length; i++)
301     {
302         var next = path[i];
303         var values = links.GetLink(current);
304         var source = values[constants.SourcePart];
305         var target = values[constants.TargetPart];
306         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
307   ↳ next))
308         {
309             //throw new InvalidOperationException(string.Format("Невозможно выбрать
310   ↳ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
311             return false;
312         }
313         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
314   ↳ target))
315         {
316             //throw new InvalidOperationException(string.Format("Невозможно продолжить
317   ↳ путь через элемент пути {0}", next));
318             return false;
319         }
320         current = next;
321     }
322     return true;
323 }
324
325 /// <remarks>
326 /// Может потребовать дополнительного стека для PathElement's при использовании
327   ↳ SequenceWalker.
328 /// </remarks>
329 [MethodImpl(MethodImplOptions.AggressiveInlining)]
330 public static TLink GetByKeyes<TLink>(this ILinks<TLink> links, TLink root, params int[]
331   ↳ path)
332 {
333     links.EnsureLinkExists(root, "root");
334     var currentLink = root;

```

```

327     for (var i = 0; i < path.Length; i++)
328     {
329         currentLink = links.GetLink(currentLink)[path[i]];
330     }
331     return currentLink;
332 }
333
334 /// <summary>
335 /// <para>
336 /// Gets the square matrix sequence element by index using the specified links.
337 /// </para>
338 /// <para></para>
339 /// </summary>
340 /// <typeparam name="TLink">
341 /// <para>The link.</para>
342 /// <para></para>
343 /// </typeparam>
344 /// <param name="links">
345 /// <para>The links.</para>
346 /// <para></para>
347 /// </param>
348 /// <param name="root">
349 /// <para>The root.</para>
350 /// <para></para>
351 /// </param>
352 /// <param name="size">
353 /// <para>The size.</para>
354 /// <para></para>
355 /// </param>
356 /// <param name="index">
357 /// <para>The index.</para>
358 /// <para></para>
359 /// </param>
360 /// <exception cref="ArgumentOutOfRangeException">
361 /// <para>Sequences with sizes other than powers of two are not supported.</para>
362 /// <para></para>
363 /// </exception>
364 /// <returns>
365 /// <para>The current link.</para>
366 /// <para></para>
367 /// </returns>
368 [MethodImpl(MethodImplOptions.AggressiveInlining)]
369 public static TLink GetSquareMatrixSequenceElementByIndex<TLink>(this ILinks<TLink>
    ↪ links, TLink root, ulong size, ulong index)
370 {
371     var constants = links.Constants;
372     var source = constants.SourcePart;
373     var target = constants.TargetPart;
374     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
375     {
376         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
    ↪ than powers of two are not supported.");
377     }
378     var path = new BitArray(BitConverter.GetBytes(index));
379     var length = Bit.GetLowestPosition(size);
380     links.EnsureLinkExists(root, "root");
381     var currentLink = root;
382     for (var i = length - 1; i >= 0; i--)
383     {
384         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
385     }
386     return currentLink;
387 }
388
389 #endregion
390
391 /// <summary>
392 /// Возвращает индекс указанной связи.
393 /// </summary>
394 /// <param name="links">Хранилище связей.</param>
395 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↪ содержимого.</param>
396 /// <returns>Индекс начальной связи для указанной связи.</returns>
397 [MethodImpl(MethodImplOptions.AggressiveInlining)]
398 public static TLink GetIndex<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
    ↪ link[links.Constants.IndexPart];
399
400 /// <summary>

```

```

401 /// Возвращает индекс начальной (Source) связи для указанной связи.
402 /// </summary>
403 /// <param name="links">Хранилище связей.</param>
404 /// <param name="link">Индекс связи.</param>
405 /// <returns>Индекс начальной связи для указанной связи.</returns>
406 [MethodImpl(MethodImplOptions.AggressiveInlining)]
407 public static TLink GetSource<TLink>(this ILinks<TLink> links, TLink link) =>
408     ↪ links.GetLink(link)[links.Constants.SourcePart];
409
410 /// <summary>
411 /// Возвращает индекс начальной (Source) связи для указанной связи.
412 /// </summary>
413 /// <param name="links">Хранилище связей.</param>
414 /// <param name="link">Связь представленная списком, состоящим из её адреса и
415     ↪ содержимого.</param>
416 /// <returns>Индекс начальной связи для указанной связи.</returns>
417 [MethodImpl(MethodImplOptions.AggressiveInlining)]
418 public static TLink GetSource<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
419     ↪ link[links.Constants.SourcePart];
420
421 /// <summary>
422 /// Возвращает индекс конечной (Target) связи для указанной связи.
423 /// </summary>
424 /// <param name="links">Хранилище связей.</param>
425 /// <param name="link">Индекс связи.</param>
426 /// <returns>Индекс конечной связи для указанной связи.</returns>
427 [MethodImpl(MethodImplOptions.AggressiveInlining)]
428 public static TLink GetTarget<TLink>(this ILinks<TLink> links, TLink link) =>
429     ↪ links.GetLink(link)[links.Constants.TargetPart];
430
431 /// <summary>
432 /// Возвращает индекс конечной (Target) связи для указанной связи.
433 /// </summary>
434 /// <param name="links">Хранилище связей.</param>
435 /// <param name="link">Связь представленная списком, состоящим из её адреса и
436     ↪ содержимого.</param>
437 /// <returns>Индекс конечной связи для указанной связи.</returns>
438 [MethodImpl(MethodImplOptions.AggressiveInlining)]
439 public static TLink GetTarget<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
440     ↪ link[links.Constants.TargetPart];
441
442 /// <summary>
443 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
444     ↪ (handler) для каждой подходящей связи.
445 /// </summary>
446 /// <param name="links">Хранилище связей.</param>
447 /// <param name="handler">Обработчик каждой подходящей связи.</param>
448 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
449     ↪ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any -
450     ↪ отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
451 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
452     ↪ случае.</returns>
453 [MethodImpl(MethodImplOptions.AggressiveInlining)]
454 public static bool Each<TLink>(this ILinks<TLink> links, ReadHandler<TLink>? handler,
455     ↪ params TLink[] restriction)
456     => EqualityComparer<TLink>.Default.Equals(links.Each(restriction, handler),
457     ↪ links.Constants.Continue);
458
459 public static bool Each<TLink>(this ILinks<TLink> links, Func<TLink, bool> handler,
460     ↪ TLink source, TLink target) => links.Each(source, target, handler);
461
462 /// <summary>
463 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
464     ↪ (handler) для каждой подходящей связи.
465 /// </summary>
466 /// <param name="links">Хранилище связей.</param>
467 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
468     ↪ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
469     ↪ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
470 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
471     ↪ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
472     ↪ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
473 /// <param name="handler">Обработчик каждой подходящей связи.</param>
474 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
475     ↪ случае.</returns>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

459 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
    ↳ Func<TLink, bool> handler)
460 {
461     var constants = links.Constants;
462     return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
    ↳ constants.Break, constants.Any, source, target);
463 }
464
465 public static bool Each<TLink>(this ILinks<TLink> links, ReadHandler<TLink>? handler,
    ↳ TLink source, TLink target) => links.Each(source, target, handler);
466
467
468 /// <summary>
469 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
470 /// </summary>
471 /// <param name="links">Хранилище связей.</param>
472 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
473 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
474 /// <param name="handler">Обработчик каждой подходящей связи.</param>
475 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
    ↳ ReadHandler<TLink>? handler) => links.Each(handler, links.Constants.Any, source,
    ↳ target);
478
479 /// <summary>
480 /// <para>
481 /// Alls the links.
482 /// </para>
483 /// <para></para>
484 /// </summary>
485 /// <typeparam name="TLink">
486 /// <para>The link.</para>
487 /// <para></para>
488 /// </typeparam>
489 /// <param name="links">
490 /// <para>The links.</para>
491 /// <para></para>
492 /// </param>
493 /// <param name="restriction">
494 /// <para>The restriction.</para>
495 /// <para></para>
496 /// </param>
497 /// <returns>
498 /// <para>A list of i list t link</para>
499 /// <para></para>
500 /// </returns>
501 [MethodImpl(MethodImplOptions.AggressiveInlining)]
502 public static IList<IList<TLink>> All<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ restriction)
503 {
504     var allLinks = new List<IList<TLink>>();
505     var filler = new ListFiller<IList<TLink>, TLink>(allLinks, links.Constants.Continue);
506     links.Each(filler.AddAndReturnConstant, restriction);
507     return allLinks;
508 }
509
510 /// <summary>
511 /// <para>
512 /// Alls the indices using the specified links.
513 /// </para>
514 /// <para></para>
515 /// </summary>
516 /// <typeparam name="TLink">
517 /// <para>The link.</para>
518 /// <para></para>
519 /// </typeparam>
520 /// <param name="links">
521 /// <para>The links.</para>
522 /// <para></para>
523 /// </param>

```

```

524 /// <param name="restriction">
525 /// <para>The restriction.</para>
526 /// <para></para>
527 /// </param>
528 /// <returns>
529 /// <para>A list of t link</para>
530 /// <para></para>
531 /// </returns>
532 [MethodImpl(MethodImplOptions.AggressiveInlining)]
533 public static IList<TLink> AllIndices<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ restriction)
534 {
535     var allIndices = new List<TLink>();
536     var filler = new ListFiller<TLink, TLink>(allIndices, links.Constants.Continue);
537     links.Each(filler.AddFirstAndReturnConstant, restriction);
538     return allIndices;
539 }
540
541 /// <summary>
542 /// Возвращает значение, определяющее существует ли связь с указанными началом и концом
    ↳ в хранилище связей.
543 /// </summary>
544 /// <param name="links">Хранилище связей.</param>
545 /// <param name="source">Начало связи.</param>
546 /// <param name="target">Конец связи.</param>
547 /// <returns>Значение, определяющее существует ли связь.</returns>
548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public static bool Exists<TLink>(this ILinks<TLink> links, TLink source, TLink target)
    ↳ => Comparer<TLink>.Default.Compare(links.Count(links.Constants.Any, source, target),
    ↳ default) > 0;
550
551 #region Ensure
552 // TODO: May be move to EnsureExtensions or make it both there and here
553
554 /// <summary>
555 /// <para>
556 /// Ensures the link exists using the specified links.
557 /// </para>
558 /// <para></para>
559 /// </summary>
560 /// <typeparam name="TLink">
561 /// <para>The link.</para>
562 /// <para></para>
563 /// </typeparam>
564 /// <param name="links">
565 /// <para>The links.</para>
566 /// <para></para>
567 /// </param>
568 /// <param name="restriction">
569 /// <para>The restriction.</para>
570 /// <para></para>
571 /// </param>
572 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
573 /// <para>sequence[{i}]</para>
574 /// <para></para>
575 /// </exception>
576 [MethodImpl(MethodImplOptions.AggressiveInlining)]
577 public static void EnsureLinkExists<TLink>(this ILinks<TLink> links, IList<TLink>
    ↳ restriction)
578 {
579     for (var i = 0; i < restriction.Count; i++)
580     {
581         if (!links.Exists(restriction[i]))
582         {
583             throw new ArgumentLinkDoesNotExistsException<TLink>(restriction[i],
                ↳ $"sequence[{i}]");
584         }
585     }
586 }
587
588 /// <summary>
589 /// <para>
590 /// Ensures the inner reference exists using the specified links.
591 /// </para>
592 /// <para></para>
593 /// </summary>
594 /// <typeparam name="TLink">
595 /// <para>The link.</para>

```

```

596 /// <para></para>
597 /// </typeparam>
598 /// <param name="links">
599 /// <para>The links.</para>
600 /// <para></para>
601 /// </param>
602 /// <param name="reference">
603 /// <para>The reference.</para>
604 /// <para></para>
605 /// </param>
606 /// <param name="argumentName">
607 /// <para>The argument name.</para>
608 /// <para></para>
609 /// </param>
610 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
611 /// <para></para>
612 /// <para></para>
613 /// </exception>
614 [MethodImpl(MethodImplOptions.AggressiveInlining)]
615 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links, TLink
↪ reference, string argumentName)
616 {
617     if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
618     {
619         throw new ArgumentLinkDoesNotExistsException<TLink>(reference, argumentName);
620     }
621 }
622
623 /// <summary>
624 /// <para>
625 /// Ensures the inner reference exists using the specified links.
626 /// </para>
627 /// <para></para>
628 /// </summary>
629 /// <typeparam name="TLink">
630 /// <para>The link.</para>
631 /// <para></para>
632 /// </typeparam>
633 /// <param name="links">
634 /// <para>The links.</para>
635 /// <para></para>
636 /// </param>
637 /// <param name="restriction">
638 /// <para>The restriction.</para>
639 /// <para></para>
640 /// </param>
641 /// <param name="argumentName">
642 /// <para>The argument name.</para>
643 /// <para></para>
644 /// </param>
645 [MethodImpl(MethodImplOptions.AggressiveInlining)]
646 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links,
↪ IList<TLink> restriction, string argumentName)
647 {
648     for (int i = 0; i < restriction.Count; i++)
649     {
650         links.EnsureInnerReferenceExists(restriction[i], argumentName);
651     }
652 }
653
654 /// <summary>
655 /// <para>
656 /// Ensures the link is any or exists using the specified links.
657 /// </para>
658 /// <para></para>
659 /// </summary>
660 /// <typeparam name="TLink">
661 /// <para>The link.</para>
662 /// <para></para>
663 /// </typeparam>
664 /// <param name="links">
665 /// <para>The links.</para>
666 /// <para></para>
667 /// </param>
668 /// <param name="restriction">
669 /// <para>The restriction.</para>
670 /// <para></para>
671 /// </param>

```



```

672 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
673 /// <para>sequence[{i}]</para>
674 /// <para></para>
675 /// </exception>
676 [MethodImpl(MethodImplOptions.AggressiveInlining)]
677 public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, IList<TLink>
↪ restriction)
678 {
679     var equalityComparer = EqualityComparer<TLink>.Default;
680     var any = links.Constants.Any;
681     for (var i = 0; i < restriction.Count; i++)
682     {
683         if (!equalityComparer.Equals(restriction[i], any) &&
↪ !links.Exists(restriction[i]))
684         {
685             throw new ArgumentLinkDoesNotExistsException<TLink>(restriction[i],
↪ $"sequence[{i}]");
686         }
687     }
688 }
689
690 /// <summary>
691 /// <para>
692 /// Ensures the link is any or exists using the specified links.
693 /// </para>
694 /// <para></para>
695 /// </summary>
696 /// <typeparam name="TLink">
697 /// <para>The link.</para>
698 /// <para></para>
699 /// </typeparam>
700 /// <param name="links">
701 /// <para>The links.</para>
702 /// <para></para>
703 /// </param>
704 /// <param name="link">
705 /// <para>The link.</para>
706 /// <para></para>
707 /// </param>
708 /// <param name="argumentName">
709 /// <para>The argument name.</para>
710 /// <para></para>
711 /// </param>
712 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
713 /// <para></para>
714 /// <para></para>
715 /// </exception>
716 [MethodImpl(MethodImplOptions.AggressiveInlining)]
717 public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, TLink link,
↪ string argumentName)
718 {
719     var equalityComparer = EqualityComparer<TLink>.Default;
720     if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
721     {
722         throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
723     }
724 }
725
726 /// <summary>
727 /// <para>
728 /// Ensures the link is itself or exists using the specified links.
729 /// </para>
730 /// <para></para>
731 /// </summary>
732 /// <typeparam name="TLink">
733 /// <para>The link.</para>
734 /// <para></para>
735 /// </typeparam>
736 /// <param name="links">
737 /// <para>The links.</para>
738 /// <para></para>
739 /// </param>
740 /// <param name="link">
741 /// <para>The link.</para>
742 /// <para></para>
743 /// </param>
744 /// <param name="argumentName">
745 /// <para>The argument name.</para>

```

```

746 /// <para></para>
747 /// </param>
748 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
749 /// <para></para>
750 /// <para></para>
751 /// </exception>
752 [MethodImpl(MethodImplOptions.AggressiveInlining)]
753 public static void EnsureLinkIsItselfOrExists<TLink>(this ILinks<TLink> links, TLink
    ↳ link, string argumentName)
754 {
755     var equalityComparer = EqualityComparer<TLink>.Default;
756     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
757     {
758         throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
759     }
760 }
761
762 /// <param name="links">Хранилище связей.</param>
763 [MethodImpl(MethodImplOptions.AggressiveInlining)]
764 public static void EnsureDoesNotExists<TLink>(this ILinks<TLink> links, TLink source,
    ↳ TLink target)
765 {
766     if (links.Exists(source, target))
767     {
768         throw new LinkWithSameValueAlreadyExistsException();
769     }
770 }
771
772 /// <param name="links">Хранилище связей.</param>
773 [MethodImpl(MethodImplOptions.AggressiveInlining)]
774 public static void EnsureNoUsages<TLink>(this ILinks<TLink> links, TLink link)
775 {
776     if (links.HasUsages(link))
777     {
778         throw new ArgumentLinkHasDependenciesException<TLink>(link);
779     }
780 }
781
782 /// <param name="links">Хранилище связей.</param>
783 [MethodImpl(MethodImplOptions.AggressiveInlining)]
784 public static void EnsureCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ addresses) => links.EnsureCreated(links.Create, addresses);
785
786 /// <param name="links">Хранилище связей.</param>
787 [MethodImpl(MethodImplOptions.AggressiveInlining)]
788 public static void EnsurePointsCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ addresses) => links.EnsureCreated(links.CreatePoint, addresses);
789
790 /// <param name="links">Хранилище связей.</param>
791 [MethodImpl(MethodImplOptions.AggressiveInlining)]
792 public static void EnsureCreated<TLink>(this ILinks<TLink> links, Func<TLink> creator,
    ↳ params TLink[] addresses)
793 {
794     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
795     var uint64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
796     var nonExistentAddresses = new HashSet<TLink>(addresses.Where(x =>
    ↳ !links.Exists(x)));
797     if (nonExistentAddresses.Count > 0)
798     {
799         var max = nonExistentAddresses.Max();
800         max = uint64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
    ↳ Convert(max),
    ↳ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
    ↳ imum)));
801         var createdLinks = new List<TLink>();
802         var equalityComparer = EqualityComparer<TLink>.Default;
803         TLink createdLink = creator();
804         while (!equalityComparer.Equals(createdLink, max))
805         {
806             createdLinks.Add(createdLink);
807         }
808         for (var i = 0; i < createdLinks.Count; i++)
809         {
810             if (!nonExistentAddresses.Contains(createdLinks[i]))
811             {
812                 links.Delete(createdLinks[i]);
813             }
814         }
815     }
816 }

```

```

815     }
816 }
817
818 #endregion
819
820 /// <param name="links">Хранилище связей.</param>
821 [MethodImpl(MethodImplOptions.AggressiveInlining)]
822 public static TLink CountUsages<TLink>(this ILinks<TLink> links, TLink link)
823 {
824     var constants = links.Constants;
825     var values = links.GetLink(link);
826     TLink usagesAsSource = links.Count(new Link<TLink>(constants.Any, link,
827         ↪ constants.Any));
828     var equalityComparer = EqualityComparer<TLink>.Default;
829     if (equalityComparer.Equals(values[constants.SourcePart], link))
830     {
831         usagesAsSource = Arithmetic<TLink>.Decrement(usagesAsSource);
832     }
833     TLink usagesAsTarget = links.Count(new Link<TLink>(constants.Any, constants.Any,
834         ↪ link));
835     if (equalityComparer.Equals(values[constants.TargetPart], link))
836     {
837         usagesAsTarget = Arithmetic<TLink>.Decrement(usagesAsTarget);
838     }
839     return Arithmetic<TLink>.Add(usagesAsSource, usagesAsTarget);
840 }
841
842 /// <param name="links">Хранилище связей.</param>
843 [MethodImpl(MethodImplOptions.AggressiveInlining)]
844 public static bool HasUsages<TLink>(this ILinks<TLink> links, TLink link) =>
845     ↪ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
846
847 /// <param name="links">Хранилище связей.</param>
848 [MethodImpl(MethodImplOptions.AggressiveInlining)]
849 public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
850     ↪ TLink target)
851 {
852     var constants = links.Constants;
853     var values = links.GetLink(link);
854     var equalityComparer = EqualityComparer<TLink>.Default;
855     return equalityComparer.Equals(values[constants.SourcePart], source) &&
856         ↪ equalityComparer.Equals(values[constants.TargetPart], target);
857 }
858
859 /// <summary>
860 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
861 /// </summary>
862 /// <param name="links">Хранилище связей.</param>
863 /// <param name="source">Индекс связи, которая является началом для искомой
864     ↪ связи.</param>
865 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
866 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
867     ↪ (концом).</returns>
868 [MethodImpl(MethodImplOptions.AggressiveInlining)]
869 public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
870     ↪ target)
871 {
872     var constants = links.Constants;
873     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break, default);
874     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
875     return setter.Result;
876 }
877
878 public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
879 {
880     var constants = links.Constants;
881     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
882     links.CreatePoint(setter.SetFirstFromSecondListAndReturnTrue);
883     return setter.Result;
884 }
885
886 /// <param name="links">Хранилище связей.</param>
887 [MethodImpl(MethodImplOptions.AggressiveInlining)]
888 public static TLink CreatePoint<TLink>(this ILinks<TLink> links, WriteHandler<TLink>
889     ↪ handler)
890 {
891     var constants = links.Constants;

```

```

883 WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
884     ↪ handler);
885 TLink link = default;
886 TLink HandlerWrapper(IList<TLink> before, IList<TLink> after)
887 {
888     link = after[constants.IndexPart];
889     return handlerState.Handler != null ? handlerState.Handler(before, after) :
890         ↪ constants.Continue;
891 }
892 handlerState.Apply(links.Create(null, HandlerWrapper));
893 handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
894 return handlerState.Result;
895 }
896
897 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
898     ↪ target)
899 {
900     var constants = links.Constants;
901     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
902     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
903     return setter.Result;
904 }
905
906 /// <param name="links">Хранилище связей.</param>
907 [MethodImpl(MethodImplOptions.AggressiveInlining)]
908 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
909     ↪ target, WriteHandler<TLink> handler)
910 {
911     var constants = links.Constants;
912     TLink createdLink = default;
913     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
914         ↪ handler);
915     handlerState.Apply(links.Create(null, (before, after) =>
916     {
917         createdLink = links.GetIndex(after);
918         return handlerState.Handler != null ? handlerState.Handler(before, after) :
919             ↪ constants.Continue;
920     })));
921     handlerState.Apply(links.Update(createdLink, source, target, handler));
922     return handlerState.Result;
923 }
924
925 /// <summary>
926 /// Обновляет связь с указанными началом (Source) и концом (Target)
927 /// на связь с указанными началом (NewSource) и концом (NewTarget).
928 /// </summary>
929 /// <param name="links">Хранилище связей.</param>
930 /// <param name="link">Индекс обновляемой связи.</param>
931 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
932     ↪ выполняется обновление.</param>
933 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
934     ↪ выполняется обновление.</param>
935 /// <returns>Индекс обновлённой связи.</returns>
936 [MethodImpl(MethodImplOptions.AggressiveInlining)]
937 public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
938     ↪ TLink newTarget) => links.Update(new LinkAddress<TLink>(link), new Link<TLink>(link,
939     ↪ newSource, newTarget));
940
941 public static TLink Update<TLink>(this ILinks<TLink> links, params TLink[] restriction)
942     ↪ => links.Update(restriction, null);
943
944 public static TLink Update<TLink>(this ILinks<TLink> links, WriteHandler<TLink> handler,
945     ↪ params TLink[] restriction) => links.Update(restriction, handler);
946
947 public static TLink Update<TLink>(this ILinks<TLink> links, IList<TLink> restriction)
948 {
949     var constants = links.Constants;
950     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
951     links.Update(restriction, setter.SetFirstFromSecondListAndReturnTrue);
952     return setter.Result;
953 }
954
955 /// <summary>
956 /// Обновляет связь с указанными началом (Source) и концом (Target)
957 /// на связь с указанными началом (NewSource) и концом (NewTarget).
958 /// </summary>

```

```

949 /// <param name="links">Хранилище связей.</param>
950 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
    ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Itself -
    ↳ требование установить ссылку на себя, 1..∞ конкретный адрес другой связи.</param>
951 /// <returns>Индекс обновлённой связи.</returns>
952 [MethodImpl(MethodImplOptions.AggressiveInlining)]
953 public static TLink Update<TLink>(this ILinks<TLink> links, IList<TLink> restriction,
    ↳ WriteHandler<TLink> handler)
954 {
955     return restriction.Count switch
956     {
957         2 => links.MergeAndDelete(restriction[0], restriction[1], handler),
958         4 => links.UpdateOrCreateOrGet(restriction[0], restriction[1], restriction[2],
            ↳ restriction[3], handler),
959         _ => links.Update(restriction[0], restriction[1], restriction[2], handler)
960     };
961 }
962
963 public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
    ↳ TLink newTarget, WriteHandler<TLink> handler) => links.Update(new
    ↳ LinkAddress<TLink>(link), new Link<TLink>(link, newSource, newTarget), handler);
964
965 /// <summary>
966 /// <para>
967 /// Resolves the constant as self reference using the specified links.
968 /// </para>
969 /// <para></para>
970 /// </summary>
971 /// <typeparam name="TLink">
972 /// <para>The link.</para>
973 /// <para></para>
974 /// </typeparam>
975 /// <param name="links">
976 /// <para>The links.</para>
977 /// <para></para>
978 /// </param>
979 /// <param name="constant">
980 /// <para>The constant.</para>
981 /// <para></para>
982 /// </param>
983 /// <param name="restriction">
984 /// <para>The restriction.</para>
985 /// <para></para>
986 /// </param>
987 /// <param name="substitution">
988 /// <para>The substitution.</para>
989 /// <para></para>
990 /// </param>
991 /// <returns>
992 /// <para>A list of t link</para>
993 /// <para></para>
994 /// </returns>
995 [MethodImpl(MethodImplOptions.AggressiveInlining)]
996 public static IList<TLink> ResolveConstantAsSelfReference<TLink>(this ILinks<TLink>
    ↳ links, TLink constant, IList<TLink> restriction, IList<TLink> substitution)
997 {
998     var equalityComparer = EqualityComparer<TLink>.Default;
999     var constants = links.Constants;
1000     var restrictionIndex = restriction[constants.IndexPart];
1001     var substitutionIndex = substitution[constants.IndexPart];
1002     if (equalityComparer.Equals(substitutionIndex, default))
1003     {
1004         substitutionIndex = restrictionIndex;
1005     }
1006     var source = substitution[constants.SourcePart];
1007     var target = substitution[constants.TargetPart];
1008     source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
1009     target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
1010     return new Link<TLink>(substitutionIndex, source, target);
1011 }
1012
1013 /// <summary>
1014 /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
    ↳ с указанными Source (началом) и Target (концом).
1015 /// </summary>
1016 /// <param name="links">Хранилище связей.</param>
1017 /// <param name="source">Индекс связи, которая является началом на создаваемой
    ↳ связи.</param>

```

```

1018 /// <param name="target">Индекс связи, которая является концом для создаваемой
1019   ↳ связи.</param>
1020 /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
1021 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1022 public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
1023   ↳ target)
1024 {
1025     var link = links.SearchOrDefault(source, target);
1026     if (EqualityComparer<TLink>.Default.Equals(link, default))
1027     {
1028         link = links.CreateAndUpdate(source, target);
1029     }
1030     return link;
1031 }
1032
1033 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1034   ↳ TLink target, TLink newSource, TLink newTarget)
1035 {
1036     var constants = links.Constants;
1037     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
1038     links.UpdateOrCreateOrGet(source, target, newSource, newTarget,
1039       ↳ setter.SetFirstFromSecondListAndReturnTrue);
1040     return setter.Result;
1041 }
1042
1043 /// <summary>
1044 /// Обновляет связь с указанными началом (Source) и концом (Target)
1045 /// на связь с указанными началом (NewSource) и концом (NewTarget).
1046 /// </summary>
1047 /// <param name="links">Хранилище связей.</param>
1048 /// <param name="source">Индекс связи, которая является началом обновляемой
1049   ↳ связи.</param>
1050 /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1051 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
1052   ↳ выполняется обновление.</param>
1053 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
1054   ↳ выполняется обновление.</param>
1055 /// <returns>Индекс обновлённой связи.</returns>
1056 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1057 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1058   ↳ TLink target, TLink newSource, TLink newTarget, WriteHandler<TLink> handler)
1059 {
1060     var equalityComparer = EqualityComparer<TLink>.Default;
1061     var link = links.SearchOrDefault(source, target);
1062     if (equalityComparer.Equals(link, default))
1063     {
1064         return links.CreateAndUpdate(newSource, newTarget, handler);
1065     }
1066     if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
1067       ↳ target))
1068     {
1069         var linkStruct = new Link<TLink>(link, source, target);
1070         return link;
1071     }
1072     return links.Update(link, newSource, newTarget, handler);
1073 }
1074
1075 /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1076 /// <param name="links">Хранилище связей.</param>
1077 /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1078 /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1079 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1080 public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
1081   ↳ target)
1082 {
1083     var link = links.SearchOrDefault(source, target);
1084     if (!EqualityComparer<TLink>.Default.Equals(link, default))
1085     {
1086         links.Delete(link);
1087         return link;
1088     }
1089     return default;
1090 }
1091
1092 /// <summary>Удаляет несколько связей.</summary>
1093 /// <param name="links">Хранилище связей.</param>
1094 /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1095 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1086 public static void DeleteMany<TLink>(this ILinks<TLink> links, IList<TLink> deletedLinks)
1087 {
1088     for (int i = 0; i < deletedLinks.Count; i++)
1089     {
1090         links.Delete(deletedLinks[i]);
1091     }
1092 }
1093
1094 public static void DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex) =>
1095     ↪ links.DeleteAllUsages(linkIndex, null);
1096
1097 /// <remarks>Before execution of this method ensure that deleted link is detached (all
1098 ↪ values - source and target are reset to null) or it might enter into infinite
1099 ↪ recursion.</remarks>
1100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1101 public static TLink DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex,
1102     ↪ WriteHandler<TLink> handler)
1103 {
1104     var constants = links.Constants;
1105     var any = constants.Any;
1106     var equalityComparer = EqualityComparer<TLink>.Default;
1107     var usagesAsSourceQuery = new Link<TLink>(any, linkIndex, any);
1108     var usagesAsTargetQuery = new Link<TLink>(any, any, linkIndex);
1109     var usages = new List<IList<TLink>>();
1110     var usagesFiller = new ListFiller<IList<TLink>, TLink>(usages, constants.Continue);
1111     links.Each(usagesFiller.AddAndReturnConstant, usagesAsSourceQuery);
1112     links.Each(usagesFiller.AddAndReturnConstant, usagesAsTargetQuery);
1113     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
1114     ↪ handler);
1115     foreach (var usage in usages)
1116     {
1117         if (equalityComparer.Equals(links.GetIndex(usage), linkIndex) ||
1118             ↪ !links.Exists(links.GetIndex(usage)))
1119         {
1120             continue;
1121         }
1122         handlerState.Apply(links.Delete(links.GetIndex(usage), handlerState.Handler));
1123     }
1124     return handlerState.Result;
1125 }
1126
1127 /// <summary>
1128 /// <para>
1129 /// Deletes the by query using the specified links.
1130 /// </para>
1131 /// </summary>
1132 /// <typeparam name="TLink">
1133 /// <para>The link.</para>
1134 /// </typeparam>
1135 /// <param name="links">
1136 /// <para>The links.</para>
1137 /// </param>
1138 /// <param name="query">
1139 /// <para>The query.</para>
1140 /// </param>
1141 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1142 public static void DeleteByQuery<TLink>(this ILinks<TLink> links, Link<TLink> query)
1143 {
1144     var queryResult = new List<TLink>();
1145     var queryResultFiller = new ListFiller<TLink, TLink>(queryResult,
1146     ↪ links.Constants.Continue);
1147     links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1148     foreach (var link in queryResult)
1149     {
1150         links.Delete(link);
1151     }
1152 }
1153
1154 // TODO: Move to Platform.Data
1155 /// <summary>
1156 /// <para>
1157 /// Determines whether are values reset.
1158 /// </para>
1159 /// </summary>

```

```

1157     /// </summary>
1158     /// <typeparam name="TLink">
1159     /// <para>The link.</para>
1160     /// <para></para>
1161     /// </typeparam>
1162     /// <param name="links">
1163     /// <para>The links.</para>
1164     /// <para></para>
1165     /// </param>
1166     /// <param name="linkIndex">
1167     /// <para>The link index.</para>
1168     /// <para></para>
1169     /// </param>
1170     /// <returns>
1171     /// <para>The bool</para>
1172     /// <para></para>
1173     /// </returns>
1174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1175     public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
1176     {
1177         var nullConstant = links.Constants.Null;
1178         var equalityComparer = EqualityComparer<TLink>.Default;
1179         var link = links.GetLink(linkIndex);
1180         for (int i = 1; i < link.Count; i++)
1181         {
1182             if (!equalityComparer.Equals(link[i], nullConstant))
1183             {
1184                 return false;
1185             }
1186         }
1187         return true;
1188     }
1189
1190     public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex) =>
1191     ↪ links.ResetValues(linkIndex, null);
1192
1193     // TODO: Create a universal version of this method in Platform.Data (with using of for
1194     ↪ loop)
1195     /// <summary>
1196     /// <para>
1197     /// Resets the values using the specified links.
1198     /// </para>
1199     /// <para></para>
1200     /// </summary>
1201     /// <typeparam name="TLink">
1202     /// <para>The link.</para>
1203     /// <para></para>
1204     /// </typeparam>
1205     /// <param name="links">
1206     /// <para>The links.</para>
1207     /// <para></para>
1208     /// </param>
1209     /// <param name="linkIndex">
1210     /// <para>The link index.</para>
1211     /// <para></para>
1212     /// </param>
1213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1214     public static TLink ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex,
1215     ↪ WriteHandler<TLink> handler)
1216     {
1217         var nullConstant = links.Constants.Null;
1218         var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
1219         return links.Update(updateRequest, handler);
1220     }
1221
1222     public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1223     ↪ => links.EnforceResetValues(linkIndex, null);
1224
1225     // TODO: Create a universal version of this method in Platform.Data (with using of for
1226     ↪ loop)
1227     /// <summary>
1228     /// <para>
1229     /// Enforces the reset values using the specified links.
1230     /// </para>
1231     /// <para></para>
1232     /// </summary>
1233     /// <typeparam name="TLink">

```



```

1230 /// <para>The link.</para>
1231 /// <para></para>
1232 /// </typeparam>
1233 /// <param name="links">
1234 /// <para>The links.</para>
1235 /// <para></para>
1236 /// </param>
1237 /// <param name="linkIndex">
1238 /// <para>The link index.</para>
1239 /// <para></para>
1240 /// </param>
1241 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1242 public static TLink EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex,
    ↪ WriteHandler<TLink> handler)
1243 {
1244     if (!links.AreValuesReset(linkIndex))
1245     {
1246         return links.ResetValues(linkIndex, handler);
1247     }
1248     return links.Constants.Continue;
1249 }
1250
1251 public static void MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
    ↪ TLink newLinkIndex) => links.MergeUsages(oldLinkIndex, newLinkIndex, null);
1252
1253 /// <summary>
1254 /// Merging two usages graphs, all children of old link moved to be children of new link
    ↪ or deleted.
1255 /// </summary>
1256 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1257 public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
    ↪ TLink newLinkIndex, WriteHandler<TLink> handler)
1258 {
1259     var equalityComparer = EqualityComparer<TLink>.Default;
1260     if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1261     {
1262         return newLinkIndex;
1263     }
1264     var constants = links.Constants;
1265     var usagesAsSource = links.All(new Link<TLink>(constants.Any, oldLinkIndex,
    ↪ constants.Any));
1266     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
    ↪ handler);
1267     for (var i = 0; i < usagesAsSource.Count; i++)
1268     {
1269         var usageAsSource = usagesAsSource[i];
1270         if (equalityComparer.Equals(usageAsSource[constants.IndexPart], oldLinkIndex))
1271         {
1272             continue;
1273         }
1274         var restriction = new LinkAddress<TLink>(usageAsSource[constants.IndexPart]);
1275         var substitution = new Link<TLink>(newLinkIndex,
    ↪ usageAsSource[constants.TargetPart]);
1276         handlerState.Apply(links.Update(restriction, substitution,
    ↪ handlerState.Handler));
1277     }
1278     var usagesAsTarget = links.All(new Link<TLink>(constants.Any, constants.Any,
    ↪ oldLinkIndex));
1279     for (var i = 0; i < usagesAsTarget.Count; i++)
1280     {
1281         var usageAsTarget = usagesAsTarget[i];
1282         if (equalityComparer.Equals(usageAsTarget[constants.IndexPart], oldLinkIndex))
1283         {
1284             continue;
1285         }
1286         var restriction = links.GetLink(usageAsTarget[constants.IndexPart]);
1287         var substitution = new Link<TLink>(usageAsTarget[constants.TargetPart],
    ↪ newLinkIndex);
1288         handlerState.Apply(links.Update(restriction, substitution,
    ↪ handlerState.Handler));
1289     }
1290     return handlerState.Result;
1291 }
1292
1293 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
    ↪ TLink newLinkIndex)
1294 {

```

```

1295     var equalityComparer = EqualityComparer<TLink>.Default;
1296     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1297     {
1298         links.MergeUsages(oldLinkIndex, newLinkIndex);
1299         links.Delete(oldLinkIndex);
1300     }
1301     return newLinkIndex;
1302 }
1303
1304 /// <summary>
1305 /// Replace one link with another (replaced link is deleted, children are updated or
1306   ↳ deleted).
1307 /// </summary>
1308 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1309 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1310   ↳ TLink newLinkIndex, WriteHandler<TLink> handler)
1311 {
1312     var equalityComparer = EqualityComparer<TLink>.Default;
1313     var constants = links.Constants;
1314     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
1315   ↳ handler);
1316     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1317     {
1318         handlerState.Apply(links.MergeUsages(oldLinkIndex, newLinkIndex,
1319   ↳ handlerState.Handler));
1320         handlerState.Apply(links.Delete(oldLinkIndex, handlerState.Handler));
1321     }
1322     return handlerState.Result;
1323 }
1324
1325 /// <summary>
1326 /// <para>
1327 /// Decorates the with automatic uniqueness and usages resolution using the specified
1328   ↳ links.
1329 /// </para>
1330 /// <para></para>
1331 /// </summary>
1332 /// <typeparam name="TLink">
1333 /// <para>The link.</para>
1334 /// <para></para>
1335 /// </typeparam>
1336 /// <param name="links">
1337 /// <para>The links.</para>
1338 /// <para></para>
1339 /// </param>
1340 /// <returns>
1341 /// <para>The links.</para>
1342 /// <para></para>
1343 /// </returns>
1344 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1345 public static ILinks<TLink>
1346   ↳ DecorateWithAutomaticUniquenessAndUsagesResolution<TLink>(this ILinks<TLink> links)
1347 {
1348     links = new LinksCascadeUsagesResolver<TLink>(links);
1349     links = new NonNullContentsLinkDeletionResolver<TLink>(links);
1350     links = new LinksCascadeUniquenessAndUsagesResolver<TLink>(links);
1351     return links;
1352 }
1353
1354 /// <summary>
1355 /// <para>
1356 /// Formats the links.
1357 /// </para>
1358 /// <para></para>
1359 /// </summary>
1360 /// <typeparam name="TLink">
1361 /// <para>The link.</para>
1362 /// <para></para>
1363 /// </typeparam>
1364 /// <param name="links">
1365 /// <para>The links.</para>
1366 /// <para></para>
1367 /// </param>
1368 /// <param name="link">
1369 /// <para>The link.</para>
1370 /// <para></para>
1371 /// </param>
1372 /// </returns>

```

```

1367     /// <para>The string</para>
1368     /// <para></para>
1369     /// </returns>
1370     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1371     public static string Format<TLink>(this ILinks<TLink> links, IList<TLink> link)
1372     {
1373         var constants = links.Constants;
1374         return $"({link[constants.IndexPart]}: {link[constants.SourcePart]}
1375             ↪ {link[constants.TargetPart]});";
1376     }
1377     /// <summary>
1378     /// <para>
1379     /// Formats the links.
1380     /// </para>
1381     /// <para></para>
1382     /// </summary>
1383     /// <typeparam name="TLink">
1384     /// <para>The link.</para>
1385     /// <para></para>
1386     /// </typeparam>
1387     /// <param name="links">
1388     /// <para>The links.</para>
1389     /// <para></para>
1390     /// </param>
1391     /// <param name="link">
1392     /// <para>The link.</para>
1393     /// <para></para>
1394     /// </param>
1395     /// <returns>
1396     /// <para>The string</para>
1397     /// <para></para>
1398     /// </returns>
1399     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1400     public static string Format<TLink>(this ILinks<TLink> links, TLink link) =>
1401         ↪ links.Format(links.GetLink(link));
1402 }

```

1.24 ./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets
4  {
5      /// <summary>
6      /// <para>
7      /// Defines the synchronized links.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     /// <seealso cref="ISynchronizedLinks<TLink, ILinks<TLink>, LinksConstants<TLink>}" />
12     /// <seealso cref="ILinks<TLink>" />
13     public interface ISynchronizedLinks<TLink> : ISynchronizedLinks<TLink, ILinks<TLink>,
14         ↪ LinksConstants<TLink>>, ILinks<TLink>
15     {
16     }

```

1.25 ./csharp/Platform.Data.Doublets/Link.cs

```

1  using Platform.Collections.Lists;
2  using Platform.Exceptions;
3  using Platform.Ranges;
4  using Platform.Singletons;
5  using System;
6  using System.Collections;
7  using System.Collections.Generic;
8  using System.Runtime.CompilerServices;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets
13 {
14     /// <summary>
15     /// Структура описывающая уникальную связь.
16     /// </summary>
17     public struct Link<TLink> : IEquatable<Link<TLink>>, IReadOnlyList<TLink>, IList<TLink>
18     {
19         /// <summary>

```

```

20    /// <para>
21    /// The link.
22    /// </para>
23    /// <para></para>
24    /// </summary>
25    public static readonly Link<TLink> Null = new Link<TLink>();
26    private static readonly LinksConstants<TLink> _constants =
27        ↳ Default<LinksConstants<TLink>>.Instance;
28    private static readonly EqualityComparer<TLink> _equalityComparer =
29        ↳ EqualityComparer<TLink>.Default;
30    private const int Length = 3;
31
32    /// <summary>
33    /// <para>
34    /// The index.
35    /// </para>
36    /// <para></para>
37    /// </summary>
38    public readonly TLink Index;
39    /// <summary>
40    /// <para>
41    /// The source.
42    /// </para>
43    /// <para></para>
44    /// </summary>
45    public readonly TLink Source;
46    /// <summary>
47    /// <para>
48    /// The target.
49    /// </para>
50    /// <para></para>
51    /// </summary>
52    public readonly TLink Target;
53
54    /// <summary>
55    /// <para>
56    /// Initializes a new <see cref="Link"/> instance.
57    /// </para>
58    /// <para></para>
59    /// </summary>
60    /// <param name="values">
61    /// <para>A values.</para>
62    /// <para></para>
63    /// </param>
64    [MethodImpl(MethodImplOptions.AggressiveInlining)]
65    public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
66        ↳ Target);
67
68    /// <summary>
69    /// <para>
70    /// Initializes a new <see cref="Link"/> instance.
71    /// </para>
72    /// <para></para>
73    /// </summary>
74    /// <param name="values">
75    /// <para>A values.</para>
76    /// <para></para>
77    /// </param>
78    [MethodImpl(MethodImplOptions.AggressiveInlining)]
79    public Link(ICollection<TLink> values) => SetValues(values, out Index, out Source, out Target);
80
81    /// <summary>
82    /// <para>
83    /// Initializes a new <see cref="Link"/> instance.
84    /// </para>
85    /// <para></para>
86    /// </summary>
87    /// <param name="other">
88    /// <para>A other.</para>
89    /// <para></para>
90    /// </param>
91    /// <exception cref="NotSupportedException">
92    /// <para></para>
93    /// <para></para>
94    /// </exception>
95    [MethodImpl(MethodImplOptions.AggressiveInlining)]
96    public Link(object other)
97    {

```

```

95         if (other is Link<TLink> otherLink)
96         {
97             SetValues(ref otherLink, out Index, out Source, out Target);
98         }
99         else if (other is IList<TLink> otherList)
100        {
101            SetValues(otherList, out Index, out Source, out Target);
102        }
103        else
104        {
105            throw new NotSupportedException();
106        }
107    }
108
109    /// <summary>
110    /// <para>
111    /// Initializes a new <see cref="Link"/> instance.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="other">
116    /// <para>A other.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
    ↪ Target);
121
122    /// <summary>
123    /// <para>
124    /// Initializes a new <see cref="Link"/> instance.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="index">
129    /// <para>A index.</para>
130    /// <para></para>
131    /// </param>
132    /// <param name="source">
133    /// <para>A source.</para>
134    /// <para></para>
135    /// </param>
136    /// <param name="target">
137    /// <para>A target.</para>
138    /// <para></para>
139    /// </param>
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public Link(TLink index, TLink source, TLink target)
142    {
143        Index = index;
144        Source = source;
145        Target = target;
146    }
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
    ↪ out TLink target)
149    {
150        index = other.Index;
151        source = other.Source;
152        target = other.Target;
153    }
154    [MethodImpl(MethodImplOptions.AggressiveInlining)]
155    private static void SetValues(IList<TLink> values, out TLink index, out TLink source,
    ↪ out TLink target)
156    {
157        if (values == null)
158        {
159            index = default;
160            source = default;
161            target = default;
162            return;
163        }
164        switch (values.Count)
165        {
166            case 3:
167                index = values[0];
168                source = values[1];
169                target = values[2];
170                break;

```

```

171         case 2:
172             index = values[0];
173             source = values[1];
174             target = default;
175             break;
176         case 1:
177             index = values[0];
178             source = default;
179             target = default;
180             break;
181         default:
182             index = default;
183             source = default;
184             target = default;
185             break;
186     }
187 }
188
189 /// <summary>
190 /// <para>
191 /// Gets the hash code.
192 /// </para>
193 /// <para></para>
194 /// </summary>
195 /// <returns>
196 /// <para>The int</para>
197 /// <para></para>
198 /// </returns>
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
201
202 /// <summary>
203 /// <para>
204 /// Determines whether this instance is null.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <returns>
209 /// <para>The bool</para>
210 /// <para></para>
211 /// </returns>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
214     && _equalityComparer.Equals(Source, _constants.Null)
215     && _equalityComparer.Equals(Target, _constants.Null);
216
217 /// <summary>
218 /// <para>
219 /// Determines whether this instance equals.
220 /// </para>
221 /// <para></para>
222 /// </summary>
223 /// <param name="other">
224 /// <para>The other.</para>
225 /// <para></para>
226 /// </param>
227 /// <returns>
228 /// <para>The bool</para>
229 /// <para></para>
230 /// </returns>
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 public override bool Equals(object other) => other is Link<TLink> &&
233     ↪ Equals((Link<TLink>)other);
234
235 /// <summary>
236 /// <para>
237 /// Determines whether this instance equals.
238 /// </para>
239 /// <para></para>
240 /// </summary>
241 /// <param name="other">
242 /// <para>The other.</para>
243 /// <para></para>
244 /// </param>
245 /// <returns>
246 /// <para>The bool</para>
247 /// <para></para>
248 /// </returns>
249 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

249 public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
250                                     && _equalityComparer.Equals(Source, other.Source)
251                                     && _equalityComparer.Equals(Target, other.Target);
252
253 /// <summary>
254 /// <para>
255 /// Returns the string using the specified index.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="index">
260 /// <para>The index.</para>
261 /// <para></para>
262 /// </param>
263 /// <param name="source">
264 /// <para>The source.</para>
265 /// <para></para>
266 /// </param>
267 /// <param name="target">
268 /// <para>The target.</para>
269 /// <para></para>
270 /// </param>
271 /// <returns>
272 /// <para>The string</para>
273 /// <para></para>
274 /// </returns>
275 [MethodImpl(MethodImplOptions.AggressiveInlining)]
276 public static string ToString(TLink index, TLink source, TLink target) => $"{index}:
    ↳ {source}->{target}";
277
278 /// <summary>
279 /// <para>
280 /// Returns the string using the specified source.
281 /// </para>
282 /// <para></para>
283 /// </summary>
284 /// <param name="source">
285 /// <para>The source.</para>
286 /// <para></para>
287 /// </param>
288 /// <param name="target">
289 /// <para>The target.</para>
290 /// <para></para>
291 /// </param>
292 /// <returns>
293 /// <para>The string</para>
294 /// <para></para>
295 /// </returns>
296 [MethodImpl(MethodImplOptions.AggressiveInlining)]
297 public static string ToString(TLink source, TLink target) => $"{source}->{target}";
298
299 [MethodImpl(MethodImplOptions.AggressiveInlining)]
300 public static implicit operator TLink[] (Link<TLink> link) => link.ToArray();
301
302 [MethodImpl(MethodImplOptions.AggressiveInlining)]
303 public static implicit operator Link<TLink> (TLink[] linkArray) => new
    ↳ Link<TLink>(linkArray);
304
305 /// <summary>
306 /// <para>
307 /// Returns the string.
308 /// </para>
309 /// <para></para>
310 /// </summary>
311 /// <returns>
312 /// <para>The string</para>
313 /// <para></para>
314 /// </returns>
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
    ↳ ToString(Source, Target) : ToString(Index, Source, Target);
317
318 #region IList
319
320 /// <summary>
321 /// <para>
322 /// Gets the count value.
323 /// </para>

```

```

324     /// <para></para>
325     /// </summary>
326     public int Count
327     {
328         [MethodImpl(MethodImplOptions.AggressiveInlining)]
329         get => Length;
330     }
331
332     /// <summary>
333     /// <para>
334     /// Gets the is read only value.
335     /// </para>
336     /// <para></para>
337     /// </summary>
338     public bool IsReadOnly
339     {
340         [MethodImpl(MethodImplOptions.AggressiveInlining)]
341         get => true;
342     }
343
344     /// <summary>
345     /// <para>
346     /// The not supported exception.
347     /// </para>
348     /// <para></para>
349     /// </summary>
350     public TLink this[int index]
351     {
352         [MethodImpl(MethodImplOptions.AggressiveInlining)]
353         get
354         {
355             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
356                 ↳ nameof(index));
357             if (index == _constants.IndexPart)
358             {
359                 return Index;
360             }
361             if (index == _constants.SourcePart)
362             {
363                 return Source;
364             }
365             if (index == _constants.TargetPart)
366             {
367                 return Target;
368             }
369             throw new NotSupportedException(); // Impossible path due to
370                 ↳ Ensure.ArgumentInRange
371         }
372         [MethodImpl(MethodImplOptions.AggressiveInlining)]
373         set => throw new NotSupportedException();
374     }
375
376     /// <summary>
377     /// <para>
378     /// Gets the enumerator.
379     /// </para>
380     /// <para></para>
381     /// </summary>
382     /// <returns>
383     /// <para>The enumerator</para>
384     /// <para></para>
385     /// </returns>
386     [MethodImpl(MethodImplOptions.AggressiveInlining)]
387     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
388
389     /// <summary>
390     /// <para>
391     /// Gets the enumerator.
392     /// </para>
393     /// <para></para>
394     /// </summary>
395     /// <returns>
396     /// <para>An enumerator of t link</para>
397     /// <para></para>
398     /// </returns>
399     [MethodImpl(MethodImplOptions.AggressiveInlining)]
400     public IEnumerator<TLink> GetEnumerator()
401     {
402         yield return Index;
403     }

```



```

401         yield return Source;
402         yield return Target;
403     }
404
405     /// <summary>
406     /// <para>
407     /// Adds the item.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="item">
412     /// <para>The item.</para>
413     /// <para></para>
414     /// </param>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     public void Add(TLink item) => throw new NotSupportedException();
417
418     /// <summary>
419     /// <para>
420     /// Clears this instance.
421     /// </para>
422     /// <para></para>
423     /// </summary>
424     [MethodImpl(MethodImplOptions.AggressiveInlining)]
425     public void Clear() => throw new NotSupportedException();
426
427     /// <summary>
428     /// <para>
429     /// Determines whether this instance contains.
430     /// </para>
431     /// <para></para>
432     /// </summary>
433     /// <param name="item">
434     /// <para>The item.</para>
435     /// <para></para>
436     /// </param>
437     /// <returns>
438     /// <para>The bool</para>
439     /// <para></para>
440     /// </returns>
441     [MethodImpl(MethodImplOptions.AggressiveInlining)]
442     public bool Contains(TLink item) => IndexOf(item) >= 0;
443
444     /// <summary>
445     /// <para>
446     /// Copies the to using the specified array.
447     /// </para>
448     /// <para></para>
449     /// </summary>
450     /// <param name="array">
451     /// <para>The array.</para>
452     /// <para></para>
453     /// </param>
454     /// <param name="arrayIndex">
455     /// <para>The array index.</para>
456     /// <para></para>
457     /// </param>
458     /// <exception cref="InvalidOperationException">
459     /// <para></para>
460     /// <para></para>
461     /// </exception>
462     [MethodImpl(MethodImplOptions.AggressiveInlining)]
463     public void CopyTo(TLink[] array, int arrayIndex)
464     {
465         Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
466         Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
467             ↪ nameof(arrayIndex));
468         if (arrayIndex + Length > array.Length)
469         {
470             throw new InvalidOperationException();
471         }
472         array[arrayIndex++] = Index;
473         array[arrayIndex++] = Source;
474         array[arrayIndex] = Target;
475     }
476
477     /// <summary>
478     /// <para>

```

```

478     /// Determines whether this instance remove.
479     /// </para>
480     /// <para></para>
481     /// </summary>
482     /// <param name="item">
483     /// <para>The item.</para>
484     /// <para></para>
485     /// </param>
486     /// <returns>
487     /// <para>The bool</para>
488     /// <para></para>
489     /// </returns>
490     [MethodImpl(MethodImplOptions.AggressiveInlining)]
491     public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
492
493     /// <summary>
494     /// <para>
495     /// Indexes the of using the specified item.
496     /// </para>
497     /// <para></para>
498     /// </summary>
499     /// <param name="item">
500     /// <para>The item.</para>
501     /// <para></para>
502     /// </param>
503     /// <returns>
504     /// <para>The int</para>
505     /// <para></para>
506     /// </returns>
507     [MethodImpl(MethodImplOptions.AggressiveInlining)]
508     public int IndexOf(TLink item)
509     {
510         if (_equalityComparer.Equals(Index, item))
511         {
512             return _constants.IndexPart;
513         }
514         if (_equalityComparer.Equals(Source, item))
515         {
516             return _constants.SourcePart;
517         }
518         if (_equalityComparer.Equals(Target, item))
519         {
520             return _constants.TargetPart;
521         }
522         return -1;
523     }
524
525     /// <summary>
526     /// <para>
527     /// Inserts the index.
528     /// </para>
529     /// <para></para>
530     /// </summary>
531     /// <param name="index">
532     /// <para>The index.</para>
533     /// <para></para>
534     /// </param>
535     /// <param name="item">
536     /// <para>The item.</para>
537     /// <para></para>
538     /// </param>
539     [MethodImpl(MethodImplOptions.AggressiveInlining)]
540     public void Insert(int index, TLink item) => throw new NotSupportedException();
541
542     /// <summary>
543     /// <para>
544     /// Removes the at using the specified index.
545     /// </para>
546     /// <para></para>
547     /// </summary>
548     /// <param name="index">
549     /// <para>The index.</para>
550     /// <para></para>
551     /// </param>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     public void RemoveAt(int index) => throw new NotSupportedException();
554
555     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

556     public static bool operator ==(Link<TLink> left, Link<TLink> right) =>
557         ↪ left.Equals(right);
558
559     [MethodImpl(MethodImplOptions.AggressiveInlining)]
560     public static bool operator !=(Link<TLink> left, Link<TLink> right) => !(left == right);
561
562     #endregion
563 }

```

1.26 ./csharp/Platform.Data.Doublets/LinkExtensions.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the link extensions.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public static class LinkExtensions
14     {
15         /// <summary>
16         /// <para>
17         /// Determines whether is full point.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         /// <typeparam name="TLink">
22         /// <para>The link.</para>
23         /// <para></para>
24         /// </typeparam>
25         /// <param name="link">
26         /// <para>The link.</para>
27         /// <para></para>
28         /// </param>
29         /// <returns>
30         /// <para>The bool</para>
31         /// <para></para>
32         /// </returns>
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public static bool IsFullPoint<TLink>(this Link<TLink> link) =>
35             ↪ Point<TLink>.IsFullPoint(link);
36
37         /// <summary>
38         /// <para>
39         /// Determines whether is partial point.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         /// <typeparam name="TLink">
44         /// <para>The link.</para>
45         /// <para></para>
46         /// </typeparam>
47         /// <param name="link">
48         /// <para>The link.</para>
49         /// <para></para>
50         /// </param>
51         /// <returns>
52         /// <para>The bool</para>
53         /// <para></para>
54         /// </returns>
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         public static bool IsPartialPoint<TLink>(this Link<TLink> link) =>
57             ↪ Point<TLink>.IsPartialPoint(link);
58     }
59 }

```

1.27 ./csharp/Platform.Data.Doublets/LinksOperatorBase.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets
6  {

```

```

7      /// <summary>
8      /// <para>
9      /// Represents the links operator base.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public abstract class LinksOperatorBase<TLink>
14     {
15         /// <summary>
16         /// <para>
17         /// The links.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         protected readonly ILinks<TLink> _links;
22
23         /// <summary>
24         /// <para>
25         /// Gets the links value.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public ILinks<TLink> Links
30         {
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             get => _links;
33         }
34
35         /// <summary>
36         /// <para>
37         /// Initializes a new <see cref="LinksOperatorBase"/> instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="links">
42         /// <para>A links.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         protected LinksOperatorBase(ILinks<TLink> links) => _links = links;
47     }
48 }

```

1.28 ./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs

```

1     using System.Runtime.CompilerServices;
2
3     #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5     namespace Platform.Data.Doublets.Memory
6     {
7         /// <summary>
8         /// <para>
9         /// Defines the links list methods.
10        /// </para>
11        /// <para></para>
12        /// </summary>
13        public interface ILinksListMethods<TLink>
14        {
15            /// <summary>
16            /// <para>
17            /// Detaches the free link.
18            /// </para>
19            /// <para></para>
20            /// </summary>
21            /// <param name="freeLink">
22            /// <para>The free link.</para>
23            /// <para></para>
24            /// </param>
25            [MethodImpl(MethodImplOptions.AggressiveInlining)]
26            void Detach(TLink freeLink);
27
28            /// <summary>
29            /// <para>
30            /// Attaches the as first using the specified link.
31            /// </para>
32            /// <para></para>
33            /// </summary>
34            /// <param name="link">
35            /// <para>The link.</para>

```

```

36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     void AttachAsFirst(TLink link);
40 }
41 }

```

1.29 ./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory
9  {
10     /// <summary>
11     /// <para>
12     /// Defines the links tree methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public interface ILinksTreeMethods<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// Counts the usages using the specified root.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="root">
25         /// <para>The root.</para>
26         /// <para></para>
27         /// </param>
28         /// <returns>
29         /// <para>The link</para>
30         /// <para></para>
31         /// </returns>
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         TLink CountUsages(TLink root);
34
35         /// <summary>
36         /// <para>
37         /// Searches the source.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="source">
42         /// <para>The source.</para>
43         /// <para></para>
44         /// </param>
45         /// <param name="target">
46         /// <para>The target.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The link</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         TLink Search(TLink source, TLink target);
55
56         /// <summary>
57         /// <para>
58         /// Eaches the usage using the specified root.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         /// <param name="root">
63         /// <para>The root.</para>
64         /// <para></para>
65         /// </param>
66         /// <param name="handler">
67         /// <para>The handler.</para>
68         /// <para></para>
69         /// </param>
70         /// <returns>
71         /// <para>The link</para>

```

```

72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     TLink EachUsage(TLink root, ReadHandler<TLink>? handler);
76
77     /// <summary>
78     /// <para>
79     /// Detaches the root.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="root">
84     /// <para>The root.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="linkIndex">
88     /// <para>The link index.</para>
89     /// <para></para>
90     /// </param>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     void Detach(ref TLink root, TLink linkIndex);
93
94     /// <summary>
95     /// <para>
96     /// Attaches the root.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="root">
101    /// <para>The root.</para>
102    /// <para></para>
103    /// </param>
104    /// <param name="linkIndex">
105    /// <para>The link index.</para>
106    /// <para></para>
107    /// </param>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    void Attach(ref TLink root, TLink linkIndex);
110 }
111 }

```

1.30 ./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets.Memory
4  {
5      /// <summary>
6      /// <para>
7      /// The index tree type enum.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public enum IndexTreeType
12     {
13         /// <summary>
14         /// <para>
15         /// The default index tree type.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         Default = 0,
20         /// <summary>
21         /// <para>
22         /// The size balanced tree index tree type.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         SizeBalancedTree = 1,
27         /// <summary>
28         /// <para>
29         /// The recursionless size balanced tree index tree type.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         RecursionlessSizeBalancedTree = 2,
34         /// <summary>
35         /// <para>
36         /// The sized and threaded avl balanced tree index tree type.

```



```

71     /// <para>
72     /// The last free link.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLink LastFreeLink;
77     /// <summary>
78     /// <para>
79     /// The reserved.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLink Reserved8;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is LinksHeader<TLink> linksHeader ?
    ↪ Equals(linksHeader) : false;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance equals.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="other">
109    /// <para>The other.</para>
110    /// <para></para>
111    /// </param>
112    /// <returns>
113    /// <para>The bool</para>
114    /// <para></para>
115    /// </returns>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    public bool Equals(LinksHeader<TLink> other)
118        => _equalityComparer.Equals(AllocatedLinks, other.AllocatedLinks)
119        && _equalityComparer.Equals(ReservedLinks, other.ReservedLinks)
120        && _equalityComparer.Equals(FreeLinks, other.FreeLinks)
121        && _equalityComparer.Equals(FirstFreeLink, other.FirstFreeLink)
122        && _equalityComparer.Equals(RootAsSource, other.RootAsSource)
123        && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
124        && _equalityComparer.Equals>LastFreeLink, other.LastFreeLink)
125        && _equalityComparer.Equals(Reserved8, other.Reserved8);
126
127    /// <summary>
128    /// <para>
129    /// Gets the hash code.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <returns>
134    /// <para>The int</para>
135    /// <para></para>
136    /// </returns>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
    ↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();
139
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public static bool operator ==(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
    ↪ left.Equals(right);
142
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator !=(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
    ↪ !(left == right);

```



```
145 }
146 }
```

1.32 ./csharp/Platform.Data.Doublents/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethod

```
1 using System;
2 using System.Text;
3 using System.Collections.Generic;
4 using System.Runtime.CompilerServices;
5 using Platform.Collections.Methods.Trees;
6 using Platform.Converters;
7 using Platform.Delegates;
8 using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublents.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the external links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}"/>
21     /// <seealso cref="ILinksTreeMethods{TLink}"/>
22     public unsafe abstract class ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
23         RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26             ↪ UncheckedConverter<TLink, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// The links data parts.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* LinksDataParts;
51
52         /// <summary>
53         /// <para>
54         /// The links index parts.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         protected readonly byte* LinksIndexParts;
59
60         /// <summary>
61         /// <para>
62         /// The header.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         protected readonly byte* Header;
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see
71         ↪ cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         /// <param name="constants">
76         /// <para>A constants.</para>
77         /// <para></para>
78         /// </param>
79         /// <param name="linksDataParts">
```

```

73     /// <para>A links data parts.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="linksIndexParts">
77     /// <para>A links index parts.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="header">
81     /// <para>A header.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
86     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
87     {
88         LinksDataParts = linksDataParts;
89         LinksIndexParts = linksIndexParts;
90         Header = header;
91         Break = constants.Break;
92         Continue = constants.Continue;
93     }
94     /// <summary>
95     /// <para>
96     /// Gets the tree root.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    protected abstract TLink GetTreeRoot();
106
107    /// <summary>
108    /// <para>
109    /// Gets the base part value using the specified link.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="link">
114    /// <para>The link.</para>
115    /// <para></para>
116    /// </param>
117    /// <returns>
118    /// <para>The link</para>
119    /// <para></para>
120    /// </returns>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    protected abstract TLink GetBasePartValue(TLink link);
123
124    /// <summary>
125    /// <para>
126    /// Determines whether this instance first is to the right of second.
127    /// </para>
128    /// <para></para>
129    /// </summary>
130    /// <param name="source">
131    /// <para>The source.</para>
132    /// <para></para>
133    /// </param>
134    /// <param name="target">
135    /// <para>The target.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="rootSource">
139    /// <para>The root source.</para>
140    /// <para></para>
141    /// </param>
142    /// <param name="rootTarget">
143    /// <para>The root target.</para>
144    /// <para></para>
145    /// </param>
146    /// <returns>
147    /// <para>The bool</para>
148    /// <para></para>
149    /// </returns>

```

```

150 [MethodImpl(MethodImplOptions.AggressiveInlining)]
151 protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);

152
153 /// <summary>
154 /// <para>
155 /// Determines whether this instance first is to the left of second.
156 /// </para>
157 /// <para></para>
158 /// </summary>
159 /// <param name="source">
160 /// <para>The source.</para>
161 /// <para></para>
162 /// </param>
163 /// <param name="target">
164 /// <para>The target.</para>
165 /// <para></para>
166 /// </param>
167 /// <param name="rootSource">
168 /// <para>The root source.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="rootTarget">
172 /// <para>The root target.</para>
173 /// <para></para>
174 /// </param>
175 /// <returns>
176 /// <para>The bool</para>
177 /// <para></para>
178 /// </returns>
179 [MethodImpl(MethodImplOptions.AggressiveInlining)]
180 protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);

181
182 /// <summary>
183 /// <para>
184 /// Gets the header reference.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 /// <returns>
189 /// <para>A ref links header of t link</para>
190 /// <para></para>
191 /// </returns>
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLink>>(Header);

194
195 /// <summary>
196 /// <para>
197 /// Gets the link data part reference using the specified link.
198 /// </para>
199 /// <para></para>
200 /// </summary>
201 /// <param name="link">
202 /// <para>The link.</para>
203 /// <para></para>
204 /// </param>
205 /// <returns>
206 /// <para>A ref raw link data part of t link</para>
207 /// <para></para>
208 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));

211
212 /// <summary>
213 /// <para>
214 /// Gets the link index part reference using the specified link.
215 /// </para>
216 /// <para></para>
217 /// </summary>
218 /// <param name="link">
219 /// <para>The link.</para>
220 /// <para></para>
221 /// </param>

```

```

222     /// <returns>
223     /// <para>A ref raw link index part of t link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
228     ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
229     ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
230
231     /// <summary>
232     /// <para>
233     /// Gets the link values using the specified link index.
234     /// </para>
235     /// <para></para>
236     /// </summary>
237     /// <param name="linkIndex">
238     /// <para>The link index.</para>
239     /// <para></para>
240     /// </param>
241     /// <returns>
242     /// <para>A list of t link</para>
243     /// <para></para>
244     /// </returns>
245     [MethodImpl(MethodImplOptions.AggressiveInlining)]
246     protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
247     {
248         ref var link = ref GetLinkDataPartReference(linkIndex);
249         return new Link<TLink>(linkIndex, link.Source, link.Target);
250     }
251
252     /// <summary>
253     /// <para>
254     /// Determines whether this instance first is to the left of second.
255     /// </para>
256     /// <para></para>
257     /// </summary>
258     /// <param name="first">
259     /// <para>The first.</para>
260     /// <para></para>
261     /// </param>
262     /// <param name="second">
263     /// <para>The second.</para>
264     /// <para></para>
265     /// </param>
266     /// <returns>
267     /// <para>The bool</para>
268     /// <para></para>
269     /// </returns>
270     [MethodImpl(MethodImplOptions.AggressiveInlining)]
271     protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
272     {
273         ref var firstLink = ref GetLinkDataPartReference(first);
274         ref var secondLink = ref GetLinkDataPartReference(second);
275         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
276         ↪ secondLink.Source, secondLink.Target);
277     }
278
279     /// <summary>
280     /// <para>
281     /// Determines whether this instance first is to the right of second.
282     /// </para>
283     /// <para></para>
284     /// </summary>
285     /// <param name="first">
286     /// <para>The first.</para>
287     /// <para></para>
288     /// </param>
289     /// <param name="second">
290     /// <para>The second.</para>
291     /// <para></para>
292     /// </param>
293     /// <returns>
294     /// <para>The bool</para>
295     /// <para></para>
296     /// </returns>
297     [MethodImpl(MethodImplOptions.AggressiveInlining)]
298     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
299     {

```

```

297     ref var firstLink = ref GetLinkDataPartReference(first);
298     ref var secondLink = ref GetLinkDataPartReference(second);
299     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
300 }
301
302 /// <summary>
303 /// <para>
304 /// The zero.
305 /// </para>
306 /// <para></para>
307 /// </summary>
308 public TLink this[TLink index]
309 {
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     get
312     {
313         var root = GetTreeRoot();
314         if (GreaterOrEqualThan(index, GetSize(root)))
315         {
316             return Zero;
317         }
318         while (!EqualToZero(root))
319         {
320             var left = GetLeftOrDefault(root);
321             var leftSize = GetSizeOrZero(left);
322             if (LessThan(index, leftSize))
323             {
324                 root = left;
325                 continue;
326             }
327             if (AreEqual(index, leftSize))
328             {
329                 return root;
330             }
331             root = GetRightOrDefault(root);
332             index = Subtract(index, Increment(leftSize));
333         }
334         return Zero; // TODO: Impossible situation exception (only if tree structure
    ↪ broken)
335     }
336 }
337
338 /// <summary>
339 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
340 /// </summary>
341 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
342 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
343 /// <returns>Индекс искомой связи.</returns>
344 [MethodImpl(MethodImplOptions.AggressiveInlining)]
345 public TLink Search(TLink source, TLink target)
346 {
347     var root = GetTreeRoot();
348     while (!EqualToZero(root))
349     {
350         ref var rootLink = ref GetLinkDataPartReference(root);
351         var rootSource = rootLink.Source;
352         var rootTarget = rootLink.Target;
353         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
    ↪ node.Key < root.Key
354         {
355             root = GetLeftOrDefault(root);
356         }
357         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
    ↪ node.Key > root.Key
358         {
359             root = GetRightOrDefault(root);
360         }
361         else // node.Key == root.Key
362         {
363             return root;
364         }
365     }
366     return Zero;
367 }
368
369 // TODO: Return indices range instead of references count

```

```

370    /// <summary>
371    /// <para>
372    /// Counts the usages using the specified link.
373    /// </para>
374    /// <para></para>
375    /// </summary>
376    /// <param name="link">
377    /// <para>The link.</para>
378    /// <para></para>
379    /// </param>
380    /// <returns>
381    /// <para>The link</para>
382    /// <para></para>
383    /// </returns>
384    [MethodImpl(MethodImplOptions.AggressiveInlining)]
385    public TLink CountUsages(TLink link)
386    {
387        var root = GetTreeRoot();
388        var total = GetSize(root);
389        var totalRightIgnore = Zero;
390        while (!EqualToZero(root))
391        {
392            var @base = GetBasePartValue(root);
393            if (LessOrEqualThan(@base, link))
394            {
395                root = GetRightOrDefault(root);
396            }
397            else
398            {
399                totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
400                root = GetLeftOrDefault(root);
401            }
402        }
403        root = GetTreeRoot();
404        var totalLeftIgnore = Zero;
405        while (!EqualToZero(root))
406        {
407            var @base = GetBasePartValue(root);
408            if (GreaterOrEqualThan(@base, link))
409            {
410                root = GetLeftOrDefault(root);
411            }
412            else
413            {
414                totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
415                root = GetRightOrDefault(root);
416            }
417        }
418        return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
419    }
420
421    /// <summary>
422    /// <para>
423    /// Eaches the usage using the specified base.
424    /// </para>
425    /// <para></para>
426    /// </summary>
427    /// <param name="@base">
428    /// <para>The base.</para>
429    /// <para></para>
430    /// </param>
431    /// <param name="handler">
432    /// <para>The handler.</para>
433    /// <para></para>
434    /// </param>
435    /// <returns>
436    /// <para>The link</para>
437    /// <para></para>
438    /// </returns>
439    [MethodImpl(MethodImplOptions.AggressiveInlining)]
440    public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
441        ↪ GetTreeRoot(), handler);
442
443    // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
444    ↪ low-level MSIL stack.
445    [MethodImpl(MethodImplOptions.AggressiveInlining)]
446    private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
447    {

```

```

446     var @continue = Continue;
447     if (EqualToZero(link))
448     {
449         return @continue;
450     }
451     var linkBasePart = GetBasePartValue(link);
452     var @break = Break;
453     if (GreaterThan(linkBasePart, @base))
454     {
455         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
456         {
457             return @break;
458         }
459     }
460     else if (LessThan(linkBasePart, @base))
461     {
462         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
463         {
464             return @break;
465         }
466     }
467     else //if (linkBasePart == @base)
468     {
469         if (AreEqual(handler(GetLinkValues(link)), @break))
470         {
471             return @break;
472         }
473         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
474         {
475             return @break;
476         }
477         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
478         {
479             return @break;
480         }
481     }
482     return @continue;
483 }
484
485 /// <summary>
486 /// <para>
487 /// Prints the node value using the specified node.
488 /// </para>
489 /// <para></para>
490 /// </summary>
491 /// <param name="node">
492 /// <para>The node.</para>
493 /// <para></para>
494 /// </param>
495 /// <param name="sb">
496 /// <para>The sb.</para>
497 /// <para></para>
498 /// </param>
499 [MethodImpl(MethodImplOptions.AggressiveInlining)]
500 protected override void PrintNodeValue(TLink node, StringBuilder sb)
501 {
502     ref var link = ref GetLinkDataPartReference(node);
503     sb.Append(' ');
504     sb.Append(link.Source);
505     sb.Append('-');
506     sb.Append('>');
507     sb.Append(link.Target);
508 }
509 }
510 }

```

1.33 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic

```

```

13 {
14     /// <summary>
15     /// <para>
16     /// Represents the external links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
21     /// <seealso cref="ILinksTreeMethods{TLink}"/>
22     public unsafe abstract class ExternalLinksSizeBalancedTreeMethodsBase<TLink> :
23     ↪ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26         ↪ UncheckedConverter<TLink, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// The links data parts.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* LinksDataParts;
51
52         /// <summary>
53         /// <para>
54         /// The links index parts.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         protected readonly byte* LinksIndexParts;
59
60         /// <summary>
61         /// <para>
62         /// The header.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         protected readonly byte* Header;
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see cref="ExternalLinksSizeBalancedTreeMethodsBase"/> instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="constants">
75         /// <para>A constants.</para>
76         /// <para></para>
77         /// </param>
78         /// <param name="linksDataParts">
79         /// <para>A links data parts.</para>
80         /// <para></para>
81         /// </param>
82         /// <param name="linksIndexParts">
83         /// <para>A links index parts.</para>
84         /// <para></para>
85         /// </param>
86         /// <param name="header">
87         /// <para>A header.</para>
88         /// <para></para>
89         /// </param>
90         [MethodImpl(MethodImplOptions.AggressiveInlining)]
91         protected ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
92         ↪ byte* linksDataParts, byte* linksIndexParts, byte* header)
93         {
94             LinksDataParts = linksDataParts;
95         }
96     }
97 }

```



```

88     LinksIndexParts = linksIndexParts;
89     Header = header;
90     Break = constants.Break;
91     Continue = constants.Continue;
92 }
93
94 /// <summary>
95 /// <para>
96 /// Gets the tree root.
97 /// </para>
98 /// <para></para>
99 /// </summary>
100 /// <returns>
101 /// <para>The link</para>
102 /// <para></para>
103 /// </returns>
104 [MethodImpl(MethodImplOptions.AggressiveInlining)]
105 protected abstract TLink GetTreeRoot();
106
107 /// <summary>
108 /// <para>
109 /// Gets the base part value using the specified link.
110 /// </para>
111 /// <para></para>
112 /// </summary>
113 /// <param name="link">
114 /// <para>The link.</para>
115 /// <para></para>
116 /// </param>
117 /// <returns>
118 /// <para>The link</para>
119 /// <para></para>
120 /// </returns>
121 [MethodImpl(MethodImplOptions.AggressiveInlining)]
122 protected abstract TLink GetBasePartValue(TLink link);
123
124 /// <summary>
125 /// <para>
126 /// Determines whether this instance first is to the right of second.
127 /// </para>
128 /// <para></para>
129 /// </summary>
130 /// <param name="source">
131 /// <para>The source.</para>
132 /// <para></para>
133 /// </param>
134 /// <param name="target">
135 /// <para>The target.</para>
136 /// <para></para>
137 /// </param>
138 /// <param name="rootSource">
139 /// <para>The root source.</para>
140 /// <para></para>
141 /// </param>
142 /// <param name="rootTarget">
143 /// <para>The root target.</para>
144 /// <para></para>
145 /// </param>
146 /// <returns>
147 /// <para>The bool</para>
148 /// <para></para>
149 /// </returns>
150 [MethodImpl(MethodImplOptions.AggressiveInlining)]
151 protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
152
153 /// <summary>
154 /// <para>
155 /// Determines whether this instance first is to the left of second.
156 /// </para>
157 /// <para></para>
158 /// </summary>
159 /// <param name="source">
160 /// <para>The source.</para>
161 /// <para></para>
162 /// </param>
163 /// <param name="target">
164 /// <para>The target.</para>

```

```

165     /// <para></para>
166     /// </param>
167     /// <param name="rootSource">
168     /// <para>The root source.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="rootTarget">
172     /// <para>The root target.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);
181
182     /// <summary>
183     /// <para>
184     /// Gets the header reference.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <returns>
189     /// <para>A ref links header of t link</para>
190     /// <para></para>
191     /// </returns>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪ AsRef<LinksHeader<TLink>>(Header);
194
195     /// <summary>
196     /// <para>
197     /// Gets the link data part reference using the specified link.
198     /// </para>
199     /// <para></para>
200     /// </summary>
201     /// <param name="link">
202     /// <para>The link.</para>
203     /// <para></para>
204     /// </param>
205     /// <returns>
206     /// <para>A ref raw link data part of t link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));
211
212     /// <summary>
213     /// <para>
214     /// Gets the link index part reference using the specified link.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="link">
219     /// <para>The link.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>A ref raw link index part of t link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
        ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
228
229     /// <summary>
230     /// <para>
231     /// Gets the link values using the specified link index.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="linkIndex">
236     /// <para>The link index.</para>

```

```

237 /// <para></para>
238 /// </param>
239 /// <returns>
240 /// <para>A list of t link</para>
241 /// <para></para>
242 /// </returns>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
245 {
246     ref var link = ref GetLinkDataPartReference(linkIndex);
247     return new Link<TLink>(linkIndex, link.Source, link.Target);
248 }
249
250 /// <summary>
251 /// <para>
252 /// Determines whether this instance first is to the left of second.
253 /// </para>
254 /// <para></para>
255 /// </summary>
256 /// <param name="first">
257 /// <para>The first.</para>
258 /// <para></para>
259 /// </param>
260 /// <param name="second">
261 /// <para>The second.</para>
262 /// <para></para>
263 /// </param>
264 /// <returns>
265 /// <para>The bool</para>
266 /// <para></para>
267 /// </returns>
268 [MethodImpl(MethodImplOptions.AggressiveInlining)]
269 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
270 {
271     ref var firstLink = ref GetLinkDataPartReference(first);
272     ref var secondLink = ref GetLinkDataPartReference(second);
273     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
274         ↪ secondLink.Source, secondLink.Target);
275 }
276
277 /// <summary>
278 /// <para>
279 /// Determines whether this instance first is to the right of second.
280 /// </para>
281 /// <para></para>
282 /// </summary>
283 /// <param name="first">
284 /// <para>The first.</para>
285 /// <para></para>
286 /// </param>
287 /// <param name="second">
288 /// <para>The second.</para>
289 /// <para></para>
290 /// </param>
291 /// <returns>
292 /// <para>The bool</para>
293 /// <para></para>
294 /// </returns>
295 [MethodImpl(MethodImplOptions.AggressiveInlining)]
296 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
297 {
298     ref var firstLink = ref GetLinkDataPartReference(first);
299     ref var secondLink = ref GetLinkDataPartReference(second);
300     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
301         ↪ secondLink.Source, secondLink.Target);
302 }
303
304 /// <summary>
305 /// <para>
306 /// The zero.
307 /// </para>
308 /// <para></para>
309 /// </summary>
310 public TLink this[TLink index]
311 {
312     [MethodImpl(MethodImplOptions.AggressiveInlining)]
313     get
314     {

```

```

313     var root = GetTreeRoot();
314     if (GreaterOrEqualThan(index, GetSize(root)))
315     {
316         return Zero;
317     }
318     while (!EqualToZero(root))
319     {
320         var left = GetLeftOrDefault(root);
321         var leftSize = GetSizeOrZero(left);
322         if (LessThan(index, leftSize))
323         {
324             root = left;
325             continue;
326         }
327         if (AreEqual(index, leftSize))
328         {
329             return root;
330         }
331         root = GetRightOrDefault(root);
332         index = Subtract(index, Increment(leftSize));
333     }
334     return Zero; // TODO: Impossible situation exception (only if tree structure
335                  ↪ broken)
336 }
337
338 /// <summary>
339 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
340 /// ↪ (концом).
341 /// </summary>
342 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
343 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
344 /// <returns>Индекс искомой связи.</returns>
345 [MethodImpl(MethodImplOptions.AggressiveInlining)]
346 public TLink Search(TLink source, TLink target)
347 {
348     var root = GetTreeRoot();
349     while (!EqualToZero(root))
350     {
351         ref var rootLink = ref GetLinkDataPartReference(root);
352         var rootSource = rootLink.Source;
353         var rootTarget = rootLink.Target;
354         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
355             ↪ node.Key < root.Key
356         {
357             root = GetLeftOrDefault(root);
358         }
359         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
360             ↪ node.Key > root.Key
361         {
362             root = GetRightOrDefault(root);
363         }
364         else // node.Key == root.Key
365         {
366             return root;
367         }
368     }
369     return Zero;
370 }
371
372 // TODO: Return indices range instead of references count
373 /// <summary>
374 /// <para>
375 /// Counts the usages using the specified link.
376 /// </para>
377 /// <para></para>
378 /// </summary>
379 /// <param name="link">
380 /// The link.</para>
381 /// </param>
382 /// <returns>
383 /// The link</para>
384 /// <para></para>
385 /// </returns>
386 [MethodImpl(MethodImplOptions.AggressiveInlining)]
387 public TLink CountUsages(TLink link)
388 {

```

```

387     var root = GetTreeRoot();
388     var total = GetSize(root);
389     var totalRightIgnore = Zero;
390     while (!EqualToZero(root))
391     {
392         var @base = GetBasePartValue(root);
393         if (LessOrEqualThan(@base, link))
394         {
395             root = GetRightOrDefault(root);
396         }
397         else
398         {
399             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
400             root = GetLeftOrDefault(root);
401         }
402     }
403     root = GetTreeRoot();
404     var totalLeftIgnore = Zero;
405     while (!EqualToZero(root))
406     {
407         var @base = GetBasePartValue(root);
408         if (GreaterOrEqualThan(@base, link))
409         {
410             root = GetLeftOrDefault(root);
411         }
412         else
413         {
414             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
415             root = GetRightOrDefault(root);
416         }
417     }
418     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
419 }
420
421 /// <summary>
422 /// <para>
423 /// Eaches the usage using the specified base.
424 /// </para>
425 /// <para></para>
426 /// </summary>
427 /// <param name="@base">
428 /// <para>The base.</para>
429 /// <para></para>
430 /// </param>
431 /// <param name="handler">
432 /// <para>The handler.</para>
433 /// <para></para>
434 /// </param>
435 /// <returns>
436 /// <para>The link</para>
437 /// <para></para>
438 /// </returns>
439 [MethodImpl(MethodImplOptions.AggressiveInlining)]
440 public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
    ↪ GetTreeRoot(), handler);
441
442 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↪ low-level MSIL stack.
443 [MethodImpl(MethodImplOptions.AggressiveInlining)]
444 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
445 {
446     var @continue = Continue;
447     if (EqualToZero(link))
448     {
449         return @continue;
450     }
451     var linkBasePart = GetBasePartValue(link);
452     var @break = Break;
453     if (GreaterThan(linkBasePart, @base))
454     {
455         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
456         {
457             return @break;
458         }
459     }
460     else if (LessThan(linkBasePart, @base))
461     {
462         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))

```

```

463         {
464             return @break;
465         }
466     }
467     else //if (linkBasePart == @base)
468     {
469         if (AreEqual(handler(GetLinkValues(link)), @break))
470         {
471             return @break;
472         }
473         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
474         {
475             return @break;
476         }
477         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
478         {
479             return @break;
480         }
481     }
482     return @continue;
483 }
484
485 /// <summary>
486 /// <para>
487 /// Prints the node value using the specified node.
488 /// </para>
489 /// <para></para>
490 /// </summary>
491 /// <param name="node">
492 /// <para>The node.</para>
493 /// <para></para>
494 /// </param>
495 /// <param name="sb">
496 /// <para>The sb.</para>
497 /// <para></para>
498 /// </param>
499 [MethodImpl(MethodImplOptions.AggressiveInlining)]
500 protected override void PrintNodeValue(TLink node, StringBuilder sb)
501 {
502     ref var link = ref GetLinkDataPartReference(node);
503     sb.Append(' ');
504     sb.Append(link.Source);
505     sb.Append('-');
506     sb.Append('>');
507     sb.Append(link.Target);
508 }
509 }
510 }

```

1.34 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTree

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the external links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
15        ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="linksDataParts">

```

```

27     /// <para>A links data parts.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="linksIndexParts">
31     /// <para>A links index parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↳ base(constants, linksDataParts, linksIndexParts, header) { }

40
41     /// <summary>
42     /// <para>
43     /// Gets the left reference using the specified node.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="node">
48     /// <para>The node.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The ref link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLink GetLeftReference(TLink node) => ref
        ↳ GetLinkIndexPartReference(node).LeftAsSource;

57
58     /// <summary>
59     /// <para>
60     /// Gets the right reference using the specified node.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="node">
65     /// <para>The node.</para>
66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>The ref link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLink GetRightReference(TLink node) => ref
        ↳ GetLinkIndexPartReference(node).RightAsSource;

74
75     /// <summary>
76     /// <para>
77     /// Gets the left using the specified node.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// <para></para>
84     /// </param>
85     /// <returns>
86     /// <para>The link</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLink GetLeft(TLink node) =>
        ↳ GetLinkIndexPartReference(node).LeftAsSource;

91
92     /// <summary>
93     /// <para>
94     /// Gets the right using the specified node.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="node">

```

```

99     /// <para>The node.</para>
100    /// <para></para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) =>
108        ↪ GetLinkIndexPartReference(node).RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↪ GetLinkIndexPartReference(node).RightAsSource = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>
157    /// <para>The link</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override TLink GetSize(TLink node) =>
162        ↪ GetLinkIndexPartReference(node).SizeAsSource;
163
164    /// <summary>
165    /// <para>
166    /// Sets the size using the specified node.
167    /// </para>
168    /// <para></para>
169    /// </summary>
170    /// <param name="node">
171    /// <para>The node.</para>
172    /// <para></para>
173    /// </param>
174    /// <param name="size">
175    /// <para>The size.</para>
176    /// <para></para>
177    /// </param>

```



```

173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected override void SetSize(TLink node, TLink size) =>
176         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
177
178     /// <summary>
179     /// <para>
180     /// Gets the tree root.
181     /// </para>
182     /// <para></para>
183     /// </summary>
184     /// <returns>
185     /// <para>The link</para>
186     /// <para></para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
190
191     /// <summary>
192     /// <para>
193     /// Gets the base part value using the specified link.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="link">
198     /// <para>The link.</para>
199     /// <para></para>
200     /// </param>
201     /// <returns>
202     /// <para>The link</para>
203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override TLink GetBasePartValue(TLink link) =>
207         ↪ GetLinkDataPartReference(link).Source;
208
209     /// <summary>
210     /// <para>
211     /// Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
237         ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
238         ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">

```

```

247     /// <para>The first target.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondSource">
251     /// <para>The second source.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondTarget">
255     /// <para>The second target.</para>
256     /// <para></para>
257     /// </param>
258     /// <returns>
259     /// <para>The bool</para>
260     /// <para></para>
261     /// </returns>
262     [MethodImpl(MethodImplOptions.AggressiveInlining)]
263     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
        ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
264
265     /// <summary>
266     /// <para>
267     /// Clears the node using the specified node.
268     /// </para>
269     /// <para></para>
270     /// </summary>
271     /// <param name="node">
272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsSource = Zero;
280         link.RightAsSource = Zero;
281         link.SizeAsSource = Zero;
282     }
283 }
284 }

```

1.35 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the external links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class ExternalLinksSourcesSizeBalancedTreeMethods<TLink> :
        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="ExternalLinksSourcesSizeBalancedTreeMethods"/> instance.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>
24        /// <para></para>
25        /// </param>
26        /// <param name="linksDataParts">
27        /// <para>A links data parts.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="linksIndexParts">
31        /// <para>A links index parts.</para>
32        /// <para></para>
33        /// </param>
34        /// <param name="header">
35        /// <para>A header.</para>

```

```

36    /// <para></para>
37    /// </param>
38    [MethodImpl(MethodImplOptions.AggressiveInlining)]
39    public ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink> constants,
    ↪    byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
    ↪    linksDataParts, linksIndexParts, header) { }

40
41    /// <summary>
42    /// <para>
43    /// Gets the left reference using the specified node.
44    /// </para>
45    /// <para></para>
46    /// </summary>
47    /// <param name="node">
48    /// <para>The node.</para>
49    /// <para></para>
50    /// </param>
51    /// <returns>
52    /// <para>The ref link</para>
53    /// <para></para>
54    /// </returns>
55    [MethodImpl(MethodImplOptions.AggressiveInlining)]
56    protected override ref TLink GetLeftReference(TLink node) => ref
    ↪    GetLinkIndexPartReference(node).LeftAsSource;

57
58    /// <summary>
59    /// <para>
60    /// Gets the right reference using the specified node.
61    /// </para>
62    /// <para></para>
63    /// </summary>
64    /// <param name="node">
65    /// <para>The node.</para>
66    /// <para></para>
67    /// </param>
68    /// <returns>
69    /// <para>The ref link</para>
70    /// <para></para>
71    /// </returns>
72    [MethodImpl(MethodImplOptions.AggressiveInlining)]
73    protected override ref TLink GetRightReference(TLink node) => ref
    ↪    GetLinkIndexPartReference(node).RightAsSource;

74
75    /// <summary>
76    /// <para>
77    /// Gets the left using the specified node.
78    /// </para>
79    /// <para></para>
80    /// </summary>
81    /// <param name="node">
82    /// <para>The node.</para>
83    /// <para></para>
84    /// </param>
85    /// <returns>
86    /// <para>The link</para>
87    /// <para></para>
88    /// </returns>
89    [MethodImpl(MethodImplOptions.AggressiveInlining)]
90    protected override TLink GetLeft(TLink node) =>
    ↪    GetLinkIndexPartReference(node).LeftAsSource;

91
92    /// <summary>
93    /// <para>
94    /// Gets the right using the specified node.
95    /// </para>
96    /// <para></para>
97    /// </summary>
98    /// <param name="node">
99    /// <para>The node.</para>
100    /// <para></para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) =>
    ↪    GetLinkIndexPartReference(node).RightAsSource;

```

```

108     /// <summary>
109     /// <para>
110     /// Sets the left using the specified node.
111     /// </para>
112     /// <para></para>
113     /// </summary>
114     /// <param name="node">
115     /// <para>The node.</para>
116     /// <para></para>
117     /// </param>
118     /// <param name="left">
119     /// <para>The left.</para>
120     /// <para></para>
121     /// </param>
122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
123     protected override void SetLeft(TLink node, TLink left) =>
124     ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126     /// <summary>
127     /// <para>
128     /// Sets the right using the specified node.
129     /// </para>
130     /// <para></para>
131     /// </summary>
132     /// <param name="node">
133     /// <para>The node.</para>
134     /// <para></para>
135     /// </param>
136     /// <param name="right">
137     /// <para>The right.</para>
138     /// <para></para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     protected override void SetRight(TLink node, TLink right) =>
142     ↪ GetLinkIndexPartReference(node).RightAsSource = right;
143
144     /// <summary>
145     /// <para>
146     /// Gets the size using the specified node.
147     /// </para>
148     /// <para></para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override TLink GetSize(TLink node) =>
160     ↪ GetLinkIndexPartReference(node).SizeAsSource;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLink node, TLink size) =>
178     ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root.
183     /// </para>
184     /// <para></para>

```

```

182    /// </summary>
183    /// <returns>
184    /// <para>The link</para>
185    /// <para></para>
186    /// </returns>
187    [MethodImpl(MethodImplOptions.AggressiveInlining)]
188    protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
189
190    /// <summary>
191    /// <para>
192    /// Gets the base part value using the specified link.
193    /// </para>
194    /// <para></para>
195    /// </summary>
196    /// <param name="link">
197    /// <para>The link.</para>
198    /// <para></para>
199    /// </param>
200    /// <returns>
201    /// <para>The link</para>
202    /// <para></para>
203    /// </returns>
204    [MethodImpl(MethodImplOptions.AggressiveInlining)]
205    protected override TLink GetBasePartValue(TLink link) =>
206        ↪ GetLinkDataPartReference(link).Source;
207
208    /// <summary>
209    /// <para>
210    /// Determines whether this instance first is to the left of second.
211    /// </para>
212    /// <para></para>
213    /// </summary>
214    /// <param name="firstSource">
215    /// <para>The first source.</para>
216    /// <para></para>
217    /// </param>
218    /// <param name="firstTarget">
219    /// <para>The first target.</para>
220    /// <para></para>
221    /// </param>
222    /// <param name="secondSource">
223    /// <para>The second source.</para>
224    /// <para></para>
225    /// </param>
226    /// <param name="secondTarget">
227    /// <para>The second target.</para>
228    /// <para></para>
229    /// </param>
230    /// <returns>
231    /// <para>The bool</para>
232    /// <para></para>
233    /// </returns>
234    [MethodImpl(MethodImplOptions.AggressiveInlining)]
235    protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
237        ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
238
239    /// <summary>
240    /// <para>
241    /// Determines whether this instance first is to the right of second.
242    /// </para>
243    /// <para></para>
244    /// </summary>
245    /// <param name="firstSource">
246    /// <para>The first source.</para>
247    /// <para></para>
248    /// </param>
249    /// <param name="firstTarget">
250    /// <para>The first target.</para>
251    /// <para></para>
252    /// </param>
253    /// <param name="secondSource">
254    /// <para>The second source.</para>
255    /// <para></para>
256    /// </param>
257    /// <param name="secondTarget">
258    /// <para>The second target.</para>
259    /// <para></para>
260    /// </param>
261    /// <returns>
262    /// <para>The bool</para>
263    /// <para></para>
264    /// </returns>

```

```

257     /// </param>
258     /// <returns>
259     /// <para>The bool</para>
260     /// <para></para>
261     /// </returns>
262     [MethodImpl(MethodImplOptions.AggressiveInlining)]
263     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
        ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
264
265     /// <summary>
266     /// <para>
267     /// Clears the node using the specified node.
268     /// </para>
269     /// <para></para>
270     /// </summary>
271     /// <param name="node">
272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsSource = Zero;
280         link.RightAsSource = Zero;
281         link.SizeAsSource = Zero;
282     }
283 }
284 }

```

1.36 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTree

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the external links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
        ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see
19        ↪ cref="ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="linksDataParts">
28        /// <para>A links data parts.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="linksIndexParts">
32        /// <para>A links index parts.</para>
33        /// <para></para>
34        /// </param>
35        /// <param name="header">
36        /// <para>A header.</para>
37        /// <para></para>
38        /// </param>
39        [MethodImpl(MethodImplOptions.AggressiveInlining)]
40        public ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↪ base(constants, linksDataParts, linksIndexParts, header) { }
41
42        /// <summary>
43        /// <para>

```

```

43     /// Gets the left reference using the specified node.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="node">
48     /// <para>The node.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The ref link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLink GetLeftReference(TLink node) => ref
57         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
75         ↪ GetLinkIndexPartReference(node).RightAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLink GetLeft(TLink node) =>
93         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
94
95     /// <summary>
96     /// <para>
97     /// Gets the right using the specified node.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="node">
102    /// <para>The node.</para>
103    /// <para></para>
104    /// </param>
105    /// <returns>
106    /// <para>The link</para>
107    /// <para></para>
108    /// </returns>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    protected override TLink GetRight(TLink node) =>
111        ↪ GetLinkIndexPartReference(node).RightAsTarget;
112
113    /// <summary>
114    /// <para>
115    /// Sets the left using the specified node.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="node">
120    /// <para>The node.</para>

```

```

117     /// <para></para>
118     /// </param>
119     /// <param name="left">
120     /// <para>The left.</para>
121     /// <para></para>
122     /// </param>
123     [MethodImpl(MethodImplOptions.AggressiveInlining)]
124     protected override void SetLeft(TLink node, TLink left) =>
125         ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
126
127     /// <summary>
128     /// <para>
129     /// Sets the right using the specified node.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↪ GetLinkIndexPartReference(node).RightAsTarget = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="node">
152     /// <para>The node.</para>
153     /// <para></para>
154     /// </param>
155     /// <returns>
156     /// <para>The link</para>
157     /// <para></para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override TLink GetSize(TLink node) =>
161         ↪ GetLinkIndexPartReference(node).SizeAsTarget;
162
163     /// <summary>
164     /// <para>
165     /// Sets the size using the specified node.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="node">
170     /// <para>The node.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="size">
174     /// <para>The size.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetSize(TLink node, TLink size) =>
179         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
180
181     /// <summary>
182     /// <para>
183     /// Gets the tree root.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <returns>
188     /// <para>The link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
193
194     /// <summary>

```



```

191    /// <para>
192    /// Gets the base part value using the specified link.
193    /// </para>
194    /// <para></para>
195    /// </summary>
196    /// <param name="link">
197    /// <para>The link.</para>
198    /// <para></para>
199    /// </param>
200    /// <returns>
201    /// <para>The link</para>
202    /// <para></para>
203    /// </returns>
204    [MethodImpl(MethodImplOptions.AggressiveInlining)]
205    protected override TLink GetBasePartValue(TLink link) =>
206        ↪ GetLinkDataPartReference(link).Target;
207
208    /// <summary>
209    /// <para>
210    /// Determines whether this instance first is to the left of second.
211    /// </para>
212    /// <para></para>
213    /// </summary>
214    /// <param name="firstSource">
215    /// <para>The first source.</para>
216    /// <para></para>
217    /// </param>
218    /// <param name="firstTarget">
219    /// <para>The first target.</para>
220    /// <para></para>
221    /// </param>
222    /// <param name="secondSource">
223    /// <para>The second source.</para>
224    /// <para></para>
225    /// </param>
226    /// <param name="secondTarget">
227    /// <para>The second target.</para>
228    /// <para></para>
229    /// </param>
230    /// <returns>
231    /// <para>The bool</para>
232    /// <para></para>
233    /// </returns>
234    [MethodImpl(MethodImplOptions.AggressiveInlining)]
235    protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
237        ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
238
239    /// <summary>
240    /// <para>
241    /// Determines whether this instance first is to the right of second.
242    /// </para>
243    /// <para></para>
244    /// </summary>
245    /// <param name="firstSource">
246    /// <para>The first source.</para>
247    /// <para></para>
248    /// </param>
249    /// <param name="firstTarget">
250    /// <para>The first target.</para>
251    /// <para></para>
252    /// </param>
253    /// <param name="secondSource">
254    /// <para>The second source.</para>
255    /// <para></para>
256    /// </param>
257    /// <param name="secondTarget">
258    /// <para>The second target.</para>
259    /// <para></para>
260    /// </param>
261    /// <returns>
262    /// <para>The bool</para>
263    /// <para></para>
264    /// </returns>
265    [MethodImpl(MethodImplOptions.AggressiveInlining)]
266    protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
267        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
268        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

```

```

264
265     /// <summary>
266     /// <para>
267     /// Clears the node using the specified node.
268     /// </para>
269     /// <para></para>
270     /// </summary>
271     /// <param name="node">
272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsTarget = Zero;
280         link.RightAsTarget = Zero;
281         link.SizeAsTarget = Zero;
282     }
283 }
284 }

```

1.37 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.Split.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the external links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class ExternalLinksTargetsSizeBalancedTreeMethods<TLink> :
15         ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
41             ↪ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
42             ↪ linksDataParts, linksIndexParts, header) { }
43
44         /// <summary>
45         /// <para>
46         /// Gets the left reference using the specified node.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="node">
51         /// <para>The node.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>The ref link</para>

```

```

53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLink GetLeftReference(TLink node) => ref
57         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// </param>
68     /// <returns>
69     /// <para>The ref link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLink GetRightReference(TLink node) => ref
74         ↪ GetLinkIndexPartReference(node).RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// </param>
85     /// <returns>
86     /// <para>The link</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLink GetLeft(TLink node) =>
91         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) =>
108        ↪ GetLinkIndexPartReference(node).RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// </param>
119    /// <param name="left">
120    /// <para>The left.</para>
121    /// </param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    protected override void SetLeft(TLink node, TLink left) =>
124        ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;

```

```

125
126    /// <summary>
127    /// <para>
128    /// Sets the right using the specified node.
129    /// </para>
130    /// <para></para>
131    /// </summary>
132    /// <param name="node">
133    /// <para>The node.</para>
134    /// <para></para>
135    /// </param>
136    /// <param name="right">
137    /// <para>The right.</para>
138    /// <para></para>
139    /// </param>
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    protected override void SetRight(TLink node, TLink right) =>
142        ↪ GetLinkIndexPartReference(node).RightAsTarget = right;
143
144    /// <summary>
145    /// <para>
146    /// Gets the size using the specified node.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="node">
151    /// <para>The node.</para>
152    /// <para></para>
153    /// </param>
154    /// <returns>
155    /// <para>The link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected override TLink GetSize(TLink node) =>
160        ↪ GetLinkIndexPartReference(node).SizeAsTarget;
161
162    /// <summary>
163    /// <para>
164    /// Sets the size using the specified node.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="node">
169    /// <para>The node.</para>
170    /// <para></para>
171    /// </param>
172    /// <param name="size">
173    /// <para>The size.</para>
174    /// <para></para>
175    /// </param>
176    [MethodImpl(MethodImplOptions.AggressiveInlining)]
177    protected override void SetSize(TLink node, TLink size) =>
178        ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
179
180    /// <summary>
181    /// <para>
182    /// Gets the tree root.
183    /// </para>
184    /// <para></para>
185    /// </summary>
186    /// <returns>
187    /// <para>The link</para>
188    /// <para></para>
189    /// </returns>
190    [MethodImpl(MethodImplOptions.AggressiveInlining)]
191    protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
192
193    /// <summary>
194    /// <para>
195    /// Gets the base part value using the specified link.
196    /// </para>
197    /// <para></para>
198    /// </summary>
199    /// <param name="link">
200    /// <para>The link.</para>
201    /// <para></para>
202    /// </param>

```

```

200    /// <returns>
201    /// <para>The link</para>
202    /// <para></para>
203    /// </returns>
204    [MethodImpl(MethodImplOptions.AggressiveInlining)]
205    protected override TLink GetBasePartValue(TLink link) =>
206        ↪ GetLinkDataPartReference(link).Target;
207
208    /// <summary>
209    /// <para>
210    /// Determines whether this instance first is to the left of second.
211    /// </para>
212    /// <para></para>
213    /// </summary>
214    /// <param name="firstSource">
215    /// <para>The first source.</para>
216    /// <para></para>
217    /// </param>
218    /// <param name="firstTarget">
219    /// <para>The first target.</para>
220    /// <para></para>
221    /// </param>
222    /// <param name="secondSource">
223    /// <para>The second source.</para>
224    /// <para></para>
225    /// </param>
226    /// <param name="secondTarget">
227    /// <para>The second target.</para>
228    /// <para></para>
229    /// </param>
230    /// <returns>
231    /// <para>The bool</para>
232    /// <para></para>
233    /// </returns>
234    [MethodImpl(MethodImplOptions.AggressiveInlining)]
235    protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
236        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
237        ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
238
239    /// <summary>
240    /// <para>
241    /// Determines whether this instance first is to the right of second.
242    /// </para>
243    /// <para></para>
244    /// </summary>
245    /// <param name="firstSource">
246    /// <para>The first source.</para>
247    /// <para></para>
248    /// </param>
249    /// <param name="firstTarget">
250    /// <para>The first target.</para>
251    /// <para></para>
252    /// </param>
253    /// <param name="secondSource">
254    /// <para>The second source.</para>
255    /// <para></para>
256    /// </param>
257    /// <param name="secondTarget">
258    /// <para>The second target.</para>
259    /// <para></para>
260    /// </param>
261    /// <returns>
262    /// <para>The bool</para>
263    /// <para></para>
264    /// </returns>
265    [MethodImpl(MethodImplOptions.AggressiveInlining)]
266    protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
267        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
268        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
269
270    /// <summary>
271    /// <para>
272    /// Clears the node using the specified node.
273    /// </para>
274    /// <para></para>
275    /// </summary>
276    /// <param name="node">

```

```

272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsTarget = Zero;
280         link.RightAsTarget = Zero;
281         link.SizeAsTarget = Zero;
282     }
283 }
284 }

```

1.38 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethod

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the internal links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}" />
21     /// <seealso cref="ILinksTreeMethods{TLink}" />
22     public unsafe abstract class InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
23         ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26             ↪ UncheckedConverter<TLink, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// The links data parts.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* LinksDataParts;
51
52         /// <summary>
53         /// <para>
54         /// The links index parts.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         protected readonly byte* LinksIndexParts;
59
60         /// <summary>
61         /// <para>
62         /// The header.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         protected readonly byte* Header;
67
68         /// <summary>
69         /// <para>

```

```

64     /// Initializes a new <see
    ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="constants">
69     /// <para>A constants.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="linksDataParts">
73     /// <para>A links data parts.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="linksIndexParts">
77     /// <para>A links index parts.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="header">
81     /// <para>A header.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
    ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
86     {
87         LinksDataParts = linksDataParts;
88         LinksIndexParts = linksIndexParts;
89         Header = header;
90         Break = constants.Break;
91         Continue = constants.Continue;
92     }
93
94     /// <summary>
95     /// <para>
96     /// Gets the tree root using the specified link.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="link">
101    /// <para>The link.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected abstract TLink GetTreeRoot(TLink link);
110
111    /// <summary>
112    /// <para>
113    /// Gets the base part value using the specified link.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="link">
118    /// <para>The link.</para>
119    /// <para></para>
120    /// </param>
121    /// <returns>
122    /// <para>The link</para>
123    /// <para></para>
124    /// </returns>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected abstract TLink GetBasePartValue(TLink link);
127
128    /// <summary>
129    /// <para>
130    /// Gets the key part value using the specified link.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="link">
135    /// <para>The link.</para>
136    /// <para></para>
137    /// </param>
138    /// <returns>
139    /// <para>The link</para>

```

```

140    /// <para></para>
141    /// </returns>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected abstract TLink GetKeyPartValue(TLink link);
144
145    /// <summary>
146    /// <para>
147    /// Gets the link data part reference using the specified link.
148    /// </para>
149    /// <para></para>
150    /// </summary>
151    /// <param name="link">
152    /// <para>The link.</para>
153    /// <para></para>
154    /// </param>
155    /// <returns>
156    /// <para>A ref raw link data part of t link</para>
157    /// <para></para>
158    /// </returns>
159    [MethodImpl(MethodImplOptions.AggressiveInlining)]
160    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));
161
162    /// <summary>
163    /// <para>
164    /// Gets the link index part reference using the specified link.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="link">
169    /// <para>The link.</para>
170    /// <para></para>
171    /// </param>
172    /// <returns>
173    /// <para>A ref raw link index part of t link</para>
174    /// <para></para>
175    /// </returns>
176    [MethodImpl(MethodImplOptions.AggressiveInlining)]
177    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
    ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
178
179    /// <summary>
180    /// <para>
181    /// Determines whether this instance first is to the left of second.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <param name="first">
186    /// <para>The first.</para>
187    /// <para></para>
188    /// </param>
189    /// <param name="second">
190    /// <para>The second.</para>
191    /// <para></para>
192    /// </param>
193    /// <returns>
194    /// <para>The bool</para>
195    /// <para></para>
196    /// </returns>
197    [MethodImpl(MethodImplOptions.AggressiveInlining)]
198    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
    ↪ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
199
200    /// <summary>
201    /// <para>
202    /// Determines whether this instance first is to the right of second.
203    /// </para>
204    /// <para></para>
205    /// </summary>
206    /// <param name="first">
207    /// <para>The first.</para>
208    /// <para></para>
209    /// </param>
210    /// <param name="second">
211    /// <para>The second.</para>
212    /// <para></para>

```



```

213     /// </param>
214     /// <returns>
215     /// <para>The bool</para>
216     /// <para></para>
217     /// </returns>
218     [MethodImpl(MethodImplOptions.AggressiveInlining)]
219     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
220         ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
221
222     /// <summary>
223     /// <para>
224     /// Gets the link values using the specified link index.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="linkIndex">
229     /// <para>The link index.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>A list of t link</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
238     {
239         ref var link = ref GetLinkDataPartReference(linkIndex);
240         return new Link<TLink>(linkIndex, link.Source, link.Target);
241     }
242
243     /// <summary>
244     /// <para>
245     /// The zero.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     public TLink this[TLink link, TLink index]
250     {
251         [MethodImpl(MethodImplOptions.AggressiveInlining)]
252         get
253         {
254             var root = GetTreeRoot(link);
255             if (GreaterOrEqualThan(index, GetSize(root)))
256             {
257                 return Zero;
258             }
259             while (!EqualToZero(root))
260             {
261                 var left = GetLeftOrDefault(root);
262                 var leftSize = GetSizeOrZero(left);
263                 if (LessThan(index, leftSize))
264                 {
265                     root = left;
266                     continue;
267                 }
268                 if (AreEqual(index, leftSize))
269                 {
270                     return root;
271                 }
272                 root = GetRightOrDefault(root);
273                 index = Subtract(index, Increment(leftSize));
274             }
275             return Zero; // TODO: Impossible situation exception (only if tree structure
276                 ↪ broken)
277         }
278     }
279
280     /// <summary>
281     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
282     ↪ (концом).
283     /// </summary>
284     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
285     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
286     /// <returns>Индекс искомой связи.</returns>
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     public abstract TLink Search(TLink source, TLink target);
289
290     /// <summary>

```

```

288     /// <para>
289     /// Searches the core using the specified root.
290     /// </para>
291     /// <para></para>
292     /// </summary>
293     /// <param name="root">
294     /// <para>The root.</para>
295     /// <para></para>
296     /// </param>
297     /// <param name="key">
298     /// <para>The key.</para>
299     /// <para></para>
300     /// </param>
301     /// <returns>
302     /// <para>The zero.</para>
303     /// <para></para>
304     /// </returns>
305     [MethodImpl(MethodImplOptions.AggressiveInlining)]
306     protected TLink SearchCore(TLink root, TLink key)
307     {
308         while (!EqualToZero(root))
309         {
310             var rootKey = GetKeyPartValue(root);
311             if (LessThan(key, rootKey) // node.Key < root.Key
312             {
313                 root = GetLeftOrDefault(root);
314             }
315             else if (GreaterThan(key, rootKey) // node.Key > root.Key
316             {
317                 root = GetRightOrDefault(root);
318             }
319             else // node.Key == root.Key
320             {
321                 return root;
322             }
323         }
324         return Zero;
325     }
326
327     // TODO: Return indices range instead of references count
328     /// <summary>
329     /// <para>
330     /// Counts the usages using the specified link.
331     /// </para>
332     /// <para></para>
333     /// </summary>
334     /// <param name="link">
335     /// <para>The link.</para>
336     /// <para></para>
337     /// </param>
338     /// <returns>
339     /// <para>The link</para>
340     /// <para></para>
341     /// </returns>
342     [MethodImpl(MethodImplOptions.AggressiveInlining)]
343     public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
344
345     /// <summary>
346     /// <para>
347     /// Eaches the usage using the specified base.
348     /// </para>
349     /// <para></para>
350     /// </summary>
351     /// <param name="@base">
352     /// <para>The base.</para>
353     /// <para></para>
354     /// </param>
355     /// <param name="handler">
356     /// <para>The handler.</para>
357     /// <para></para>
358     /// </param>
359     /// <returns>
360     /// <para>The link</para>
361     /// <para></para>
362     /// </returns>
363     [MethodImpl(MethodImplOptions.AggressiveInlining)]
364     public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
        ↪ GetTreeRoot(@base), handler);

```

```

365 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
366 ↪ low-level MSIL stack.
367 [MethodImpl(MethodImplOptions.AggressiveInlining)]
368 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
369 {
370     var @continue = Continue;
371     if (EqualToZero(link))
372     {
373         return @continue;
374     }
375     var @break = Break;
376     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
377     {
378         return @break;
379     }
380     if (AreEqual(handler(GetLinkValues(link)), @break))
381     {
382         return @break;
383     }
384     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
385     {
386         return @break;
387     }
388     return @continue;
389 }
390
391 /// <summary>
392 /// <para>
393 /// Prints the node value using the specified node.
394 /// </para>
395 /// <para></para>
396 /// </summary>
397 /// <param name="node">
398 /// <para>The node.</para>
399 /// <para></para>
400 /// </param>
401 /// <param name="sb">
402 /// <para>The sb.</para>
403 /// <para></para>
404 /// </param>
405 [MethodImpl(MethodImplOptions.AggressiveInlining)]
406 protected override void PrintNodeValue(TLink node, StringBuilder sb)
407 {
408     ref var link = ref GetLinkDataPartReference(node);
409     sb.Append(' ');
410     sb.Append(link.Source);
411     sb.Append(' - ');
412     sb.Append(' > ');
413     sb.Append(link.Target);
414 }
415 }
416 }

```

1.39 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs

```

1 using System;
2 using System.Text;
3 using System.Collections.Generic;
4 using System.Runtime.CompilerServices;
5 using Platform.Collections.Methods.Trees;
6 using Platform.Converters;
7 using Platform.Delegates;
8 using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the internal links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLink}">
21     /// <seealso cref="ILinksTreeMethods{TLink}">
22     public unsafe abstract class InternalLinksSizeBalancedTreeMethodsBase<TLink> :
23     ↪ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

24 private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
    ↳ UncheckedConverter<TLink, long>.Default;
25
26 /// <summary>
27 /// <para>
28 /// The break.
29 /// </para>
30 /// <para></para>
31 /// </summary>
32 protected readonly TLink Break;
33 /// <summary>
34 /// <para>
35 /// The continue.
36 /// </para>
37 /// <para></para>
38 /// </summary>
39 protected readonly TLink Continue;
40 /// <summary>
41 /// <para>
42 /// The links data parts.
43 /// </para>
44 /// <para></para>
45 /// </summary>
46 protected readonly byte* LinksDataParts;
47 /// <summary>
48 /// <para>
49 /// The links index parts.
50 /// </para>
51 /// <para></para>
52 /// </summary>
53 protected readonly byte* LinksIndexParts;
54 /// <summary>
55 /// <para>
56 /// The header.
57 /// </para>
58 /// <para></para>
59 /// </summary>
60 protected readonly byte* Header;
61
62 /// <summary>
63 /// <para>
64 /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
65 /// </para>
66 /// <para></para>
67 /// </summary>
68 /// <param name="constants">
69 /// <para>A constants.</para>
70 /// <para></para>
71 /// </param>
72 /// <param name="linksDataParts">
73 /// <para>A links data parts.</para>
74 /// <para></para>
75 /// </param>
76 /// <param name="linksIndexParts">
77 /// <para>A links index parts.</para>
78 /// <para></para>
79 /// </param>
80 /// <param name="header">
81 /// <para>A header.</para>
82 /// <para></para>
83 /// </param>
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
    ↳ byte* linksDataParts, byte* linksIndexParts, byte* header)
86 {
87     LinksDataParts = linksDataParts;
88     LinksIndexParts = linksIndexParts;
89     Header = header;
90     Break = constants.Break;
91     Continue = constants.Continue;
92 }
93
94 /// <summary>
95 /// <para>
96 /// Gets the tree root using the specified link.
97 /// </para>
98 /// <para></para>
99 /// </summary>
100 /// <param name="link">

```

```

101    /// <para>The link.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected abstract TLink GetTreeRoot(TLink link);
110
111    /// <summary>
112    /// <para>
113    /// Gets the base part value using the specified link.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="link">
118    /// <para>The link.</para>
119    /// <para></para>
120    /// </param>
121    /// <returns>
122    /// <para>The link</para>
123    /// <para></para>
124    /// </returns>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected abstract TLink GetBasePartValue(TLink link);
127
128    /// <summary>
129    /// <para>
130    /// Gets the key part value using the specified link.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="link">
135    /// <para>The link.</para>
136    /// <para></para>
137    /// </param>
138    /// <returns>
139    /// <para>The link</para>
140    /// <para></para>
141    /// </returns>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected abstract TLink GetKeyPartValue(TLink link);
144
145    /// <summary>
146    /// <para>
147    /// Gets the link data part reference using the specified link.
148    /// </para>
149    /// <para></para>
150    /// </summary>
151    /// <param name="link">
152    /// <para>The link.</para>
153    /// <para></para>
154    /// </param>
155    /// <returns>
156    /// <para>A ref raw link data part of t link</para>
157    /// <para></para>
158    /// </returns>
159    [MethodImpl(MethodImplOptions.AggressiveInlining)]
160    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));
161
162    /// <summary>
163    /// <para>
164    /// Gets the link index part reference using the specified link.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="link">
169    /// <para>The link.</para>
170    /// <para></para>
171    /// </param>
172    /// <returns>
173    /// <para>A ref raw link index part of t link</para>
174    /// <para></para>
175    /// </returns>
176    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

177     protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
178         ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
179         ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance first is to the left of second.
184     /// </para>
185     /// </summary>
186     /// <param name="first">
187     /// <para>The first.</para>
188     /// </param>
189     /// <param name="second">
190     /// <para>The second.</para>
191     /// </param>
192     /// <returns>
193     /// <para>The bool</para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
197         ↪ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance first is to the right of second.
202     /// </para>
203     /// </summary>
204     /// <param name="first">
205     /// <para>The first.</para>
206     /// </param>
207     /// <param name="second">
208     /// <para>The second.</para>
209     /// </param>
210     /// <returns>
211     /// <para>The bool</para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
215         ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
216
217     /// <summary>
218     /// <para>
219     /// Gets the link values using the specified link index.
220     /// </para>
221     /// </summary>
222     /// <param name="linkIndex">
223     /// <para>The link index.</para>
224     /// </param>
225     /// <returns>
226     /// <para>A list of t link</para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
230     {
231         ref var link = ref GetLinkDataPartReference(linkIndex);
232         return new Link<TLink>(linkIndex, link.Source, link.Target);
233     }
234
235     /// <summary>
236     /// <para>
237     /// The zero.
238     /// </para>
239     /// </summary>
240     public TLink this[TLink link, TLink index]
241     {
242         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

251     get
252     {
253         var root = GetTreeRoot(link);
254         if (GreaterOrEqualThan(index, GetSize(root)))
255         {
256             return Zero;
257         }
258         while (!EqualToZero(root))
259         {
260             var left = GetLeftOrDefault(root);
261             var leftSize = GetSizeOrZero(left);
262             if (LessThan(index, leftSize))
263             {
264                 root = left;
265                 continue;
266             }
267             if (AreEqual(index, leftSize))
268             {
269                 return root;
270             }
271             root = GetRightOrDefault(root);
272             index = Subtract(index, Increment(leftSize));
273         }
274         return Zero; // TODO: Impossible situation exception (only if tree structure
275                       ↪ broken)
276     }
277
278     /// <summary>
279     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
280     /// ↪ (концом).
281     /// </summary>
282     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
283     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
284     /// <returns>Индекс искомой связи.</returns>
285     [MethodImpl(MethodImplOptions.AggressiveInlining)]
286     public abstract TLink Search(TLink source, TLink target);
287
288     /// <summary>
289     /// <para>
290     /// Searches the core using the specified root.
291     /// </para>
292     /// <para></para>
293     /// </summary>
294     /// <param name="root">
295     /// <para>The root.</para>
296     /// <para></para>
297     /// </param>
298     /// <param name="key">
299     /// <para>The key.</para>
300     /// <para></para>
301     /// </param>
302     /// <returns>
303     /// <para>The zero.</para>
304     /// <para></para>
305     /// </returns>
306     [MethodImpl(MethodImplOptions.AggressiveInlining)]
307     protected TLink SearchCore(TLink root, TLink key)
308     {
309         while (!EqualToZero(root))
310         {
311             var rootKey = GetKeyPartValue(root);
312             if (LessThan(key, rootKey)) // node.Key < root.Key
313             {
314                 root = GetLeftOrDefault(root);
315             }
316             else if (GreaterThan(key, rootKey)) // node.Key > root.Key
317             {
318                 root = GetRightOrDefault(root);
319             }
320             else // node.Key == root.Key
321             {
322                 return root;
323             }
324         }
325         return Zero;
326     }

```

```

327 // TODO: Return indices range instead of references count
328 /// <summary>
329 /// <para>
330 /// Counts the usages using the specified link.
331 /// </para>
332 /// <para></para>
333 /// </summary>
334 /// <param name="link">
335 /// <para>The link.</para>
336 /// <para></para>
337 /// </param>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
344
345 /// <summary>
346 /// <para>
347 /// Eaches the usage using the specified base.
348 /// </para>
349 /// <para></para>
350 /// </summary>
351 /// <param name="@base">
352 /// <para>The base.</para>
353 /// <para></para>
354 /// </param>
355 /// <param name="handler">
356 /// <para>The handler.</para>
357 /// <para></para>
358 /// </param>
359 /// <returns>
360 /// <para>The link</para>
361 /// <para></para>
362 /// </returns>
363 [MethodImpl(MethodImplOptions.AggressiveInlining)]
364 public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
    ↳ GetTreeRoot(@base), handler);
365
366 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↳ low-level MSIL stack.
367 [MethodImpl(MethodImplOptions.AggressiveInlining)]
368 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
369 {
370     var @continue = Continue;
371     if (EqualToZero(link))
372     {
373         return @continue;
374     }
375     var @break = Break;
376     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
377     {
378         return @break;
379     }
380     if (AreEqual(handler(GetLinkValues(link)), @break))
381     {
382         return @break;
383     }
384     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
385     {
386         return @break;
387     }
388     return @continue;
389 }
390
391 /// <summary>
392 /// <para>
393 /// Prints the node value using the specified node.
394 /// </para>
395 /// <para></para>
396 /// </summary>
397 /// <param name="node">
398 /// <para>The node.</para>
399 /// <para></para>
400 /// </param>
401 /// <param name="sb">
402 /// <para>The sb.</para>

```



```

403     /// <para></para>
404     /// </param>
405     [MethodImpl(MethodImplOptions.AggressiveInlining)]
406     protected override void PrintNodeValue(TLink node, StringBuilder sb)
407     {
408         ref var link = ref GetLinkDataPartReference(node);
409         sb.Append(' ');
410         sb.Append(link.Source);
411         sb.Append('-');
412         sb.Append('>');
413         sb.Append(link.Target);
414     }
415 }
416 }

```

1.40 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections.Methods.Lists;
5  using Platform.Converters;
6  using Platform.Delegates;
7  using static System.Runtime.CompilerServices.Unsafe;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Memory.Split.Generic
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the internal links sources linked list methods.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="RelativeCircularDoublyLinkedListMethods{TLink}"/>
20     public unsafe class InternalLinksSourcesLinkedListMethods<TLink> :
21         ↳ RelativeCircularDoublyLinkedListMethods<TLink>
22     {
23         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
24             ↳ UncheckedConverter<TLink, long>.Default;
25         private readonly byte* _linksDataParts;
26         private readonly byte* _linksIndexParts;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="constants">
51         /// <para>A constants.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="linksDataParts">
55         /// <para>A links data parts.</para>
56         /// <para></para>
57         /// </param>
58         /// <param name="linksIndexParts">
59         /// <para>A links index parts.</para>
60         /// <para></para>
61         /// </param>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
64             ↳ linksDataParts, byte* linksIndexParts)
65         {

```

```

61     _linksDataParts = linksDataParts;
62     _linksIndexParts = linksIndexParts;
63     Break = constants.Break;
64     Continue = constants.Continue;
65 }
66
67 /// <summary>
68 /// <para>
69 /// Gets the link data part reference using the specified link.
70 /// </para>
71 /// <para></para>
72 /// </summary>
73 /// <param name="link">
74 /// <para>The link.</para>
75 /// <para></para>
76 /// </param>
77 /// <returns>
78 /// <para>A ref raw link data part of t link</para>
79 /// <para></para>
80 /// </returns>
81 [MethodImpl(MethodImplOptions.AggressiveInlining)]
82 protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↳ AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
    ↳ * _addressToInt64Converter.Convert(link)));
83
84 /// <summary>
85 /// <para>
86 /// Gets the link index part reference using the specified link.
87 /// </para>
88 /// <para></para>
89 /// </summary>
90 /// <param name="link">
91 /// <para>The link.</para>
92 /// <para></para>
93 /// </param>
94 /// <returns>
95 /// <para>A ref raw link index part of t link</para>
96 /// <para></para>
97 /// </returns>
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↳ ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
    ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
100
101 /// <summary>
102 /// <para>
103 /// Gets the first using the specified head.
104 /// </para>
105 /// <para></para>
106 /// </summary>
107 /// <param name="head">
108 /// <para>The head.</para>
109 /// <para></para>
110 /// </param>
111 /// <returns>
112 /// <para>The link</para>
113 /// <para></para>
114 /// </returns>
115 [MethodImpl(MethodImplOptions.AggressiveInlining)]
116 protected override TLink GetFirst(TLink head) =>
    ↳ GetLinkIndexPartReference(head).RootAsSource;
117
118 /// <summary>
119 /// <para>
120 /// Gets the last using the specified head.
121 /// </para>
122 /// <para></para>
123 /// </summary>
124 /// <param name="head">
125 /// <para>The head.</para>
126 /// <para></para>
127 /// </param>
128 /// <returns>
129 /// <para>The link</para>
130 /// <para></para>
131 /// </returns>
132 [MethodImpl(MethodImplOptions.AggressiveInlining)]
133 protected override TLink GetLast(TLink head)

```

```

134 {
135     var first = GetLinkIndexPartReference(head).RootAsSource;
136     if (EqualToZero(first))
137     {
138         return first;
139     }
140     else
141     {
142         return GetPrevious(first);
143     }
144 }
145
146 /// <summary>
147 /// <para>
148 /// Gets the previous using the specified element.
149 /// </para>
150 /// <para></para>
151 /// </summary>
152 /// <param name="element">
153 /// <para>The element.</para>
154 /// <para></para>
155 /// </param>
156 /// <returns>
157 /// <para>The link</para>
158 /// <para></para>
159 /// </returns>
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 protected override TLink GetPrevious(TLink element) =>
162     ↪ GetLinkIndexPartReference(element).LeftAsSource;
163
164 /// <summary>
165 /// <para>
166 /// Gets the next using the specified element.
167 /// </para>
168 /// <para></para>
169 /// </summary>
170 /// <param name="element">
171 /// <para>The element.</para>
172 /// <para></para>
173 /// </param>
174 /// <returns>
175 /// <para>The link</para>
176 /// <para></para>
177 /// </returns>
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 protected override TLink GetNext(TLink element) =>
180     ↪ GetLinkIndexPartReference(element).RightAsSource;
181
182 /// <summary>
183 /// <para>
184 /// Gets the size using the specified head.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 /// <param name="head">
189 /// <para>The head.</para>
190 /// <para></para>
191 /// </param>
192 /// <returns>
193 /// <para>The link</para>
194 /// <para></para>
195 /// </returns>
196 [MethodImpl(MethodImplOptions.AggressiveInlining)]
197 protected override TLink GetSize(TLink head) =>
198     ↪ GetLinkIndexPartReference(head).SizeAsSource;
199
200 /// <summary>
201 /// <para>
202 /// Sets the first using the specified head.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="head">
207 /// <para>The head.</para>
208 /// <para></para>
209 /// </param>
210 /// <param name="element">
211 /// <para>The element.</para>

```

```

209    /// <para></para>
210    /// </param>
211    [MethodImpl(MethodImplOptions.AggressiveInlining)]
212    protected override void SetFirst(TLink head, TLink element) =>
213        ↪ GetLinkIndexPartReference(head).RootAsSource = element;
214
215    /// <summary>
216    /// <para>
217    /// Sets the last using the specified head.
218    /// </para>
219    /// </summary>
220    /// <param name="head">
221    /// <para>The head.</para>
222    /// </para></param>
223    /// <param name="element">
224    /// <para>The element.</para>
225    /// </para></param>
226    [MethodImpl(MethodImplOptions.AggressiveInlining)]
227    protected override void SetLast(TLink head, TLink element)
228    {
229        //var first = GetLinkIndexPartReference(head).RootAsSource;
230        //if (EqualToZero(first))
231        //{
232        //    SetFirst(head, element);
233        //}
234        //else
235        //{
236        //    SetPrevious(first, element);
237        //}
238    }
239
240    /// <summary>
241    /// <para>
242    /// Sets the previous using the specified element.
243    /// </para>
244    /// </summary>
245    /// <param name="element">
246    /// <para>The element.</para>
247    /// </para></param>
248    /// <param name="previous">
249    /// <para>The previous.</para>
250    /// </para></param>
251    [MethodImpl(MethodImplOptions.AggressiveInlining)]
252    protected override void SetPrevious(TLink element, TLink previous) =>
253        ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;
254
255    /// <summary>
256    /// <para>
257    /// Sets the next using the specified element.
258    /// </para>
259    /// </summary>
260    /// <param name="element">
261    /// <para>The element.</para>
262    /// </para></param>
263    /// <param name="next">
264    /// <para>The next.</para>
265    /// </para></param>
266    [MethodImpl(MethodImplOptions.AggressiveInlining)]
267    protected override void SetNext(TLink element, TLink next) =>
268        ↪ GetLinkIndexPartReference(element).RightAsSource = next;
269
270    /// <summary>
271    /// <para>
272    /// Sets the size using the specified head.
273    /// </para>
274    /// </summary>
275    /// <param name="head">
276    /// <para>The head.</para>

```

```

284    /// <para></para>
285    /// </param>
286    /// <param name="size">
287    /// <para>The size.</para>
288    /// <para></para>
289    /// </param>
290    [MethodImpl(MethodImplOptions.AggressiveInlining)]
291    protected override void SetSize(TLink head, TLink size) =>
292        ↪ GetLinkIndexPartReference(head).SizeAsSource = size;
293
294    /// <summary>
295    /// <para>
296    /// Counts the usages using the specified head.
297    /// </para>
298    /// </summary>
299    /// <param name="head">
300    /// <para>The head.</para>
301    /// <para></para>
302    /// </param>
303    /// <returns>
304    /// <para>The link</para>
305    /// <para></para>
306    /// </returns>
307    [MethodImpl(MethodImplOptions.AggressiveInlining)]
308    public TLink CountUsages(TLink head) => GetSize(head);
309
310    /// <summary>
311    /// <para>
312    /// Gets the link values using the specified link index.
313    /// </para>
314    /// <para></para>
315    /// </summary>
316    /// <param name="linkIndex">
317    /// <para>The link index.</para>
318    /// <para></para>
319    /// </param>
320    /// <returns>
321    /// <para>A list of t link</para>
322    /// <para></para>
323    /// </returns>
324    [MethodImpl(MethodImplOptions.AggressiveInlining)]
325    protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
326    {
327        ref var link = ref GetLinkDataPartReference(linkIndex);
328        return new Link<TLink>(linkIndex, link.Source, link.Target);
329    }
330
331    /// <summary>
332    /// <para>
333    /// Eaches the usage using the specified source.
334    /// </para>
335    /// <para></para>
336    /// </summary>
337    /// <param name="source">
338    /// <para>The source.</para>
339    /// <para></para>
340    /// </param>
341    /// <param name="handler">
342    /// <para>The handler.</para>
343    /// <para></para>
344    /// </param>
345    /// <returns>
346    /// <para>The continue.</para>
347    /// <para></para>
348    /// </returns>
349    [MethodImpl(MethodImplOptions.AggressiveInlining)]
350    public TLink EachUsage(TLink source, ReadHandler<TLink>? handler)
351    {
352        var @continue = Continue;
353        var @break = Break;
354        var current = GetFirst(source);
355        var first = current;
356        while (!EqualToZero(current))
357        {
358            if (AreEqual(handler(GetLinkValues(current)), @break))
359            {
360                return @break;

```

```

361         }
362         current = GetNext(current);
363         if (AreEqual(current, first))
364         {
365             return @continue;
366         }
367     }
368     return @continue;
369 }
370 }
371 }

```

1.41 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTree

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the internal links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
15        ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="linksIndexParts">
33        /// <para>A links index parts.</para>
34        /// <para></para>
35        /// </param>
36        /// <param name="header">
37        /// <para>A header.</para>
38        /// <para></para>
39        /// </param>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42            ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
43            ↪ base(constants, linksDataParts, linksIndexParts, header) { }
44
45        /// <summary>
46        /// <para>
47        /// Gets the left reference using the specified node.
48        /// </para>
49        /// <para></para>
50        /// </summary>
51        /// <param name="node">
52        /// <para>The node.</para>
53        /// <para></para>
54        /// </param>
55        /// <returns>
56        /// <para>The ref link</para>
57        /// <para></para>
58        /// </returns>
59        [MethodImpl(MethodImplOptions.AggressiveInlining)]
60        protected override ref TLink GetLeftReference(TLink node) => ref
        ↪ GetLinkIndexPartReference(node).LeftAsSource;

```

```

61    /// </para>
62    /// <para></para>
63    /// </summary>
64    /// <param name="node">
65    /// <para>The node.</para>
66    /// <para></para>
67    /// </param>
68    /// <returns>
69    /// <para>The ref link</para>
70    /// <para></para>
71    /// </returns>
72    [MethodImpl(MethodImplOptions.AggressiveInlining)]
73    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ GetLinkIndexPartReference(node).RightAsSource;
74
75    /// <summary>
76    /// <para>
77    /// Gets the left using the specified node.
78    /// </para>
79    /// <para></para>
80    /// </summary>
81    /// <param name="node">
82    /// <para>The node.</para>
83    /// <para></para>
84    /// </param>
85    /// <returns>
86    /// <para>The link</para>
87    /// <para></para>
88    /// </returns>
89    [MethodImpl(MethodImplOptions.AggressiveInlining)]
90    protected override TLink GetLeft(TLink node) =>
    ↪ GetLinkIndexPartReference(node).LeftAsSource;
91
92    /// <summary>
93    /// <para>
94    /// Gets the right using the specified node.
95    /// </para>
96    /// <para></para>
97    /// </summary>
98    /// <param name="node">
99    /// <para>The node.</para>
100    /// <para></para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) =>
    ↪ GetLinkIndexPartReference(node).RightAsSource;
108
109    /// <summary>
110    /// <para>
111    /// Sets the left using the specified node.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="node">
116    /// <para>The node.</para>
117    /// <para></para>
118    /// </param>
119    /// <param name="left">
120    /// <para>The left.</para>
121    /// <para></para>
122    /// </param>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLink node, TLink left) =>
    ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126    /// <summary>
127    /// <para>
128    /// Sets the right using the specified node.
129    /// </para>
130    /// <para></para>
131    /// </summary>
132    /// <param name="node">
133    /// <para>The node.</para>
134    /// <para></para>

```

```

135     /// </param>
136     /// <param name="right">
137     /// <para>The right.</para>
138     /// <para></para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     protected override void SetRight(TLink node, TLink right) =>
142         ↪ GetLinkIndexPartReference(node).RightAsSource = right;
143
144     /// <summary>
145     /// <para>
146     /// Gets the size using the specified node.
147     /// </para>
148     /// <para></para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override TLink GetSize(TLink node) =>
160         ↪ GetLinkIndexPartReference(node).SizeAsSource;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLink node, TLink size) =>
178         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root using the specified link.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <param name="link">
187     /// <para>The link.</para>
188     /// <para></para>
189     /// </param>
190     /// <returns>
191     /// <para>The link</para>
192     /// <para></para>
193     /// </returns>
194     [MethodImpl(MethodImplOptions.AggressiveInlining)]
195     protected override TLink GetTreeRoot(TLink link) =>
196         ↪ GetLinkIndexPartReference(link).RootAsSource;
197
198     /// <summary>
199     /// <para>
200     /// Gets the base part value using the specified link.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <param name="link">
205     /// <para>The link.</para>
206     /// <para></para>
207     /// </param>
208     /// <returns>
209     /// <para>The link</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

209     protected override TLink GetBasePartValue(TLink link) =>
210         ↪ GetLinkDataPartReference(link).Source;
211
212     /// <summary>
213     /// <para>
214     /// Gets the key part value using the specified link.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="link">
219     /// <para>The link.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLink GetKeyPartValue(TLink link) =>
228         ↪ GetLinkDataPartReference(link).Target;
229
230     /// <summary>
231     /// <para>
232     /// Clears the node using the specified node.
233     /// </para>
234     /// <para></para>
235     /// </summary>
236     /// <param name="node">
237     /// <para>The node.</para>
238     /// <para></para>
239     /// </param>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     protected override void ClearNode(TLink node)
242     {
243         ref var link = ref GetLinkIndexPartReference(node);
244         link.LeftAsSource = Zero;
245         link.RightAsSource = Zero;
246         link.SizeAsSource = Zero;
247     }
248
249     /// <summary>
250     /// <para>
251     /// Searches the source.
252     /// </para>
253     /// <para></para>
254     /// </summary>
255     /// <param name="source">
256     /// <para>The source.</para>
257     /// <para></para>
258     /// </param>
259     /// <param name="target">
260     /// <para>The target.</para>
261     /// <para></para>
262     /// </param>
263     /// <returns>
264     /// <para>The link</para>
265     /// <para></para>
266     /// </returns>
267     public override TLink Search(TLink source, TLink target) =>
268         ↪ SearchCore(GetTreeRoot(source), target);
269 }

```

1.42 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the internal links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class InternalLinksSourcesSizeBalancedTreeMethods<TLink> :
15        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>

```

```

15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="InternalLinksSourcesSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="linksDataParts">
27     /// <para>A links data parts.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="linksIndexParts">
31     /// <para>A links index parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink> constants,
40         ↳ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
41         ↳ linksDataParts, linksIndexParts, header) { }
42
43     /// <summary>
44     /// <para>
45     /// Gets the left reference using the specified node.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="node">
50     /// <para>The node.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The ref link</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override ref TLink GetLeftReference(TLink node) => ref
59         ↳ GetLinkIndexPartReference(node).LeftAsSource;
60
61     /// <summary>
62     /// <para>
63     /// Gets the right reference using the specified node.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <param name="node">
68     /// <para>The node.</para>
69     /// <para></para>
70     /// </param>
71     /// <returns>
72     /// <para>The ref link</para>
73     /// <para></para>
74     /// </returns>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     protected override ref TLink GetRightReference(TLink node) => ref
77         ↳ GetLinkIndexPartReference(node).RightAsSource;
78
79     /// <summary>
80     /// <para>
81     /// Gets the left using the specified node.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <param name="node">
86     /// <para>The node.</para>
87     /// <para></para>
88     /// </param>
89     /// <returns>
90     /// <para>The link</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     protected override TLink GetLeftReference(TLink node) =>
95         ↳ GetLinkIndexPartReference(node).LeftAsSource;
96
97     /// <summary>
98     /// <para>
99     /// Gets the right using the specified node.
100    /// </para>
101    /// <para></para>
102    /// </summary>
103    /// <param name="node">
104    /// <para>The node.</para>
105    /// <para></para>
106    /// </param>
107    /// <returns>
108    /// <para>The link</para>
109    /// <para></para>
110    /// </returns>
111    [MethodImpl(MethodImplOptions.AggressiveInlining)]
112    protected override TLink GetRightReference(TLink node) =>
113        ↳ GetLinkIndexPartReference(node).RightAsSource;
114
115     /// <summary>
116     /// <para>
117     /// Gets the left using the specified node.
118     /// </para>
119     /// <para></para>
120     /// </summary>
121     /// <param name="node">
122     /// <para>The node.</para>
123     /// <para></para>
124     /// </param>
125     /// <returns>
126     /// <para>The link</para>
127     /// <para></para>
128     /// </returns>
129     [MethodImpl(MethodImplOptions.AggressiveInlining)]
130     protected TLink GetLeftReference(TLink node) =>
131         ↳ GetLinkIndexPartReference(node).LeftAsSource;
132
133     /// <summary>
134     /// <para>
135     /// Gets the right using the specified node.
136     /// </para>
137     /// <para></para>
138     /// </summary>
139     /// <param name="node">
140     /// <para>The node.</para>
141     /// <para></para>
142     /// </param>
143     /// <returns>
144     /// <para>The link</para>
145     /// <para></para>
146     /// </returns>
147     [MethodImpl(MethodImplOptions.AggressiveInlining)]
148     protected TLink GetRightReference(TLink node) =>
149         ↳ GetLinkIndexPartReference(node).RightAsSource;
150 }

```

```

88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLink GetLeft(TLink node) =>
91         ↪ GetLinkIndexPartReference(node).LeftAsSource;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// </summary>
98     /// <param name="node">
99     /// <para>The node.</para>
100    /// </para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// </para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) =>
108        ↪ GetLinkIndexPartReference(node).RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// </summary>
115    /// <param name="node">
116    /// <para>The node.</para>
117    /// </para>
118    /// </param>
119    /// <param name="left">
120    /// <para>The left.</para>
121    /// </para>
122    /// </param>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLink node, TLink left) =>
125        ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// </summary>
132    /// <param name="node">
133    /// <para>The node.</para>
134    /// </para>
135    /// </param>
136    /// <param name="right">
137    /// <para>The right.</para>
138    /// </para>
139    /// </param>
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    protected override void SetRight(TLink node, TLink right) =>
142        ↪ GetLinkIndexPartReference(node).RightAsSource = right;
143
144    /// <summary>
145    /// <para>
146    /// Gets the size using the specified node.
147    /// </para>
148    /// </summary>
149    /// <param name="node">
150    /// <para>The node.</para>
151    /// </para>
152    /// </param>
153    /// <returns>
154    /// <para>The link</para>
155    /// </para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override TLink GetSize(TLink node) =>
159        ↪ GetLinkIndexPartReference(node).SizeAsSource;

```

```

160    /// <summary>
161    /// <para>
162    /// Sets the size using the specified node.
163    /// </para>
164    /// <para></para>
165    /// </summary>
166    /// <param name="node">
167    /// <para>The node.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="size">
171    /// <para>The size.</para>
172    /// <para></para>
173    /// </param>
174    [MethodImpl(MethodImplOptions.AggressiveInlining)]
175    protected override void SetSize(TLink node, TLink size) =>
176        ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
177
178    /// <summary>
179    /// <para>
180    /// Gets the tree root using the specified link.
181    /// </para>
182    /// <para></para>
183    /// </summary>
184    /// <param name="link">
185    /// <para>The link.</para>
186    /// <para></para>
187    /// </param>
188    /// <returns>
189    /// <para>The link</para>
190    /// <para></para>
191    /// </returns>
192    [MethodImpl(MethodImplOptions.AggressiveInlining)]
193    protected override TLink GetTreeRoot(TLink link) =>
194        ↪ GetLinkIndexPartReference(link).RootAsSource;
195
196    /// <summary>
197    /// <para>
198    /// Gets the base part value using the specified link.
199    /// </para>
200    /// <para></para>
201    /// </summary>
202    /// <param name="link">
203    /// <para>The link.</para>
204    /// <para></para>
205    /// </param>
206    /// <returns>
207    /// <para>The link</para>
208    /// <para></para>
209    /// </returns>
210    [MethodImpl(MethodImplOptions.AggressiveInlining)]
211    protected override TLink GetBasePartValue(TLink link) =>
212        ↪ GetLinkDataPartReference(link).Source;
213
214    /// <summary>
215    /// <para>
216    /// Gets the key part value using the specified link.
217    /// </para>
218    /// <para></para>
219    /// </summary>
220    /// <param name="link">
221    /// <para>The link.</para>
222    /// <para></para>
223    /// </param>
224    /// <returns>
225    /// <para>The link</para>
226    /// <para></para>
227    /// </returns>
228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
229    protected override TLink GetKeyPartValue(TLink link) =>
230        ↪ GetLinkDataPartReference(link).Target;
231
232    /// <summary>
233    /// <para>
234    /// Clears the node using the specified node.
235    /// </para>
236    /// <para></para>
237    /// </summary>

```

```

234     /// <param name="node">
235     /// <para>The node.</para>
236     /// <para></para>
237     /// </param>
238     [MethodImpl(MethodImplOptions.AggressiveInlining)]
239     protected override void ClearNode(TLink node)
240     {
241         ref var link = ref GetLinkIndexPartReference(node);
242         link.LeftAsSource = Zero;
243         link.RightAsSource = Zero;
244         link.SizeAsSource = Zero;
245     }
246
247     /// <summary>
248     /// <para>
249     /// Searches the source.
250     /// </para>
251     /// <para></para>
252     /// </summary>
253     /// <param name="source">
254     /// <para>The source.</para>
255     /// <para></para>
256     /// </param>
257     /// <param name="target">
258     /// <para>The target.</para>
259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The link</para>
263     /// <para></para>
264     /// </returns>
265     public override TLink Search(TLink source, TLink target) =>
266         ↪ SearchCore(GetTreeRoot(source), target);
267 }

```

1.43 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTree

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the internal links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
15        ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="linksIndexParts">
33        /// <para>A links index parts.</para>
34        /// <para></para>
35        /// </param>
36        /// <param name="header">
37        /// <para>A header.</para>
38        /// <para></para>
39        /// </param>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

39 public InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
    ↳ base(constants, linksDataParts, linksIndexParts, header) { }
40
41 /// <summary>
42 /// <para>
43 /// Gets the left reference using the specified node.
44 /// </para>
45 /// <para></para>
46 /// </summary>
47 /// <param name="node">
48 /// <para>The node.</para>
49 /// <para></para>
50 /// </param>
51 /// <returns>
52 /// <para>The ref link</para>
53 /// <para></para>
54 /// </returns>
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ GetLinkIndexPartReference(node).LeftAsTarget;
57
58 /// <summary>
59 /// <para>
60 /// Gets the right reference using the specified node.
61 /// </para>
62 /// <para></para>
63 /// </summary>
64 /// <param name="node">
65 /// <para>The node.</para>
66 /// <para></para>
67 /// </param>
68 /// <returns>
69 /// <para>The ref link</para>
70 /// <para></para>
71 /// </returns>
72 [MethodImpl(MethodImplOptions.AggressiveInlining)]
73 protected override ref TLink GetRightReference(TLink node) => ref
    ↳ GetLinkIndexPartReference(node).RightAsTarget;
74
75 /// <summary>
76 /// <para>
77 /// Gets the left using the specified node.
78 /// </para>
79 /// <para></para>
80 /// </summary>
81 /// <param name="node">
82 /// <para>The node.</para>
83 /// <para></para>
84 /// </param>
85 /// <returns>
86 /// <para>The link</para>
87 /// <para></para>
88 /// </returns>
89 [MethodImpl(MethodImplOptions.AggressiveInlining)]
90 protected override TLink GetLeft(TLink node) =>
    ↳ GetLinkIndexPartReference(node).LeftAsTarget;
91
92 /// <summary>
93 /// <para>
94 /// Gets the right using the specified node.
95 /// </para>
96 /// <para></para>
97 /// </summary>
98 /// <param name="node">
99 /// <para>The node.</para>
100 /// <para></para>
101 /// </param>
102 /// <returns>
103 /// <para>The link</para>
104 /// <para></para>
105 /// </returns>
106 [MethodImpl(MethodImplOptions.AggressiveInlining)]
107 protected override TLink GetRight(TLink node) =>
    ↳ GetLinkIndexPartReference(node).RightAsTarget;
108
109 /// <summary>

```

```

110    /// <para>
111    /// Sets the left using the specified node.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="node">
116    /// <para>The node.</para>
117    /// <para></para>
118    /// </param>
119    /// <param name="left">
120    /// <para>The left.</para>
121    /// <para></para>
122    /// </param>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLink node, TLink left) =>
125        ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">
134    /// <para>The node.</para>
135    /// <para></para>
136    /// </param>
137    /// <param name="right">
138    /// <para>The right.</para>
139    /// <para></para>
140    /// </param>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override void SetRight(TLink node, TLink right) =>
143        ↪ GetLinkIndexPartReference(node).RightAsTarget = right;
144
145    /// <summary>
146    /// <para>
147    /// Gets the size using the specified node.
148    /// </para>
149    /// <para></para>
150    /// </summary>
151    /// <param name="node">
152    /// <para>The node.</para>
153    /// <para></para>
154    /// </param>
155    /// <returns>
156    /// <para>The link</para>
157    /// <para></para>
158    /// </returns>
159    [MethodImpl(MethodImplOptions.AggressiveInlining)]
160    protected override TLink GetSize(TLink node) =>
161        ↪ GetLinkIndexPartReference(node).SizeAsTarget;
162
163    /// <summary>
164    /// <para>
165    /// Sets the size using the specified node.
166    /// </para>
167    /// <para></para>
168    /// </summary>
169    /// <param name="node">
170    /// <para>The node.</para>
171    /// <para></para>
172    /// </param>
173    /// <param name="size">
174    /// <para>The size.</para>
175    /// <para></para>
176    /// </param>
177    [MethodImpl(MethodImplOptions.AggressiveInlining)]
178    protected override void SetSize(TLink node, TLink size) =>
179        ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
180
181    /// <summary>
182    /// <para>
183    /// Gets the tree root using the specified link.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <param name="link">

```

```

184    /// <para>The link.</para>
185    /// <para></para>
186    /// </param>
187    /// <returns>
188    /// <para>The link</para>
189    /// <para></para>
190    /// </returns>
191    [MethodImpl(MethodImplOptions.AggressiveInlining)]
192    protected override TLink GetTreeRoot(TLink link) =>
193        ↪ GetLinkIndexPartReference(link).RootAsTarget;
194
195    /// <summary>
196    /// <para>
197    /// Gets the base part value using the specified link.
198    /// </para>
199    /// <para></para>
200    /// </summary>
201    /// <param name="link">
202    /// <para>The link.</para>
203    /// <para></para>
204    /// </param>
205    /// <returns>
206    /// <para>The link</para>
207    /// <para></para>
208    /// </returns>
209    [MethodImpl(MethodImplOptions.AggressiveInlining)]
210    protected override TLink GetBasePartValue(TLink link) =>
211        ↪ GetLinkDataPartReference(link).Target;
212
213    /// <summary>
214    /// <para>
215    /// Gets the key part value using the specified link.
216    /// </para>
217    /// <para></para>
218    /// </summary>
219    /// <param name="link">
220    /// <para>The link.</para>
221    /// <para></para>
222    /// </param>
223    /// <returns>
224    /// <para>The link</para>
225    /// <para></para>
226    /// </returns>
227    [MethodImpl(MethodImplOptions.AggressiveInlining)]
228    protected override TLink GetKeyPartValue(TLink link) =>
229        ↪ GetLinkDataPartReference(link).Source;
230
231    /// <summary>
232    /// <para>
233    /// Clears the node using the specified node.
234    /// </para>
235    /// <para></para>
236    /// </summary>
237    /// <param name="node">
238    /// <para>The node.</para>
239    /// <para></para>
240    /// </param>
241    [MethodImpl(MethodImplOptions.AggressiveInlining)]
242    protected override void ClearNode(TLink node)
243    {
244        ref var link = ref GetLinkIndexPartReference(node);
245        link.LeftAsTarget = Zero;
246        link.RightAsTarget = Zero;
247        link.SizeAsTarget = Zero;
248    }
249
250    /// <summary>
251    /// <para>
252    /// Searches the source.
253    /// </para>
254    /// <para></para>
255    /// </summary>
256    /// <param name="source">
257    /// <para>The source.</para>
258    /// <para></para>
259    /// </param>
260    /// <param name="target">
261    /// <para>The target.</para>

```



```

259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The link</para>
263     /// <para></para>
264     /// </returns>
265     public override TLink Search(TLink source, TLink target) =>
        ↪ SearchCore(GetTreeRoot(target), source);
266 }
267 }

```

1.44 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.Split.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the internal links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}" />
14     public unsafe class InternalLinksTargetsSizeBalancedTreeMethods<TLink> :
        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="InternalLinksTargetsSizeBalancedTreeMethods" /> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="linksDataParts">
27         /// <para>A links data parts.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="linksIndexParts">
31         /// <para>A links index parts.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="header">
35         /// <para>A header.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
            ↪ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
            ↪ linksDataParts, linksIndexParts, header) { }
40
41         /// <summary>
42         /// <para>
43         /// Gets the left reference using the specified node.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         /// <param name="node">
48         /// <para>The node.</para>
49         /// <para></para>
50         /// </param>
51         /// <returns>
52         /// <para>The ref link</para>
53         /// <para></para>
54         /// </returns>
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         protected override ref TLink GetLeftReference(TLink node) => ref
            ↪ GetLinkIndexPartReference(node).LeftAsTarget;
57
58         /// <summary>
59         /// <para>
60         /// Gets the right reference using the specified node.
61         /// </para>
62         /// <para></para>

```

```

63     /// </summary>
64     /// <param name="node">
65     /// <para>The node.</para>
66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>The ref link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLink GetRightReference(TLink node) => ref
74     ↪ GetLinkIndexPartReference(node).RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) =>
92     ↪ GetLinkIndexPartReference(node).LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) =>
110    ↪ GetLinkIndexPartReference(node).RightAsTarget;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLink node, TLink left) =>
128    ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">

```

```

137     /// <para>The right.</para>
138     /// <para></para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     protected override void SetRight(TLink node, TLink right) =>
142         ↪ GetLinkIndexPartReference(node).RightAsTarget = right;
143
144     /// <summary>
145     /// <para>
146     /// Gets the size using the specified node.
147     /// </para>
148     /// </summary>
149     /// <param name="node">
150     /// <para>The node.</para>
151     /// </param>
152     /// <returns>
153     /// <para>The link</para>
154     /// </returns>
155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
156     protected override TLink GetSize(TLink node) =>
157         ↪ GetLinkIndexPartReference(node).SizeAsTarget;
158
159     /// <summary>
160     /// <para>
161     /// Sets the size using the specified node.
162     /// </para>
163     /// </summary>
164     /// <param name="node">
165     /// <para>The node.</para>
166     /// </param>
167     /// <param name="size">
168     /// <para>The size.</para>
169     /// </param>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override void SetSize(TLink node, TLink size) =>
172         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
173
174     /// <summary>
175     /// <para>
176     /// Gets the tree root using the specified link.
177     /// </para>
178     /// </summary>
179     /// <param name="link">
180     /// <para>The link.</para>
181     /// </param>
182     /// <returns>
183     /// <para>The link</para>
184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected override TLink GetTreeRoot(TLink link) =>
187         ↪ GetLinkIndexPartReference(link).RootAsTarget;
188
189     /// <summary>
190     /// <para>
191     /// Gets the base part value using the specified link.
192     /// </para>
193     /// </summary>
194     /// <param name="link">
195     /// <para>The link.</para>
196     /// </param>
197     /// <returns>
198     /// <para>The link</para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override TLink GetBasePartValue(TLink link) =>
202         ↪ GetLinkDataPartReference(link).Target;

```

```

210
211     /// <summary>
212     /// <para>
213     /// Gets the key part value using the specified link.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="link">
218     /// <para>The link.</para>
219     /// <para></para>
220     /// </param>
221     /// <returns>
222     /// <para>The link</para>
223     /// <para></para>
224     /// </returns>
225     [MethodImpl(MethodImplOptions.AggressiveInlining)]
226     protected override TLink GetKeyPartValue(TLink link) =>
227         ↪ GetLinkDataPartReference(link).Source;
228
229     /// <summary>
230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref GetLinkIndexPartReference(node);
243         link.LeftAsTarget = Zero;
244         link.RightAsTarget = Zero;
245         link.SizeAsTarget = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(target), source);

```

1.45 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.Split.Generic
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the split memory links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="SplitMemoryLinksBase{TLink}"/>

```

```

18 public unsafe class SplitMemoryLinks<TLink> : SplitMemoryLinksBase<TLink>
19 {
20     private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
21     private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
22     private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
23     private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
24     private byte* _header;
25     private byte* _linksDataParts;
26     private byte* _linksIndexParts;
27
28     /// <summary>
29     /// <para>
30     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     /// <param name="dataMemory">
35     /// <para>A data memory.</para>
36     /// <para></para>
37     /// </param>
38     /// <param name="indexMemory">
39     /// <para>A index memory.</para>
40     /// <para></para>
41     /// </param>
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     public SplitMemoryLinks(string dataMemory, string indexMemory) : this(new
44         ↪ FileMappedResizableDirectMemory(dataMemory), new
45         ↪ FileMappedResizableDirectMemory(indexMemory)) { }
46
47     /// <summary>
48     /// <para>
49     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="dataMemory">
54     /// <para>A data memory.</para>
55     /// <para></para>
56     /// </param>
57     /// <param name="indexMemory">
58     /// <para>A index memory.</para>
59     /// <para></para>
60     /// </param>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
63         ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
64
65     /// <summary>
66     /// <para>
67     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="dataMemory">
72     /// <para>A data memory.</para>
73     /// <para></para>
74     /// </param>
75     /// <param name="indexMemory">
76     /// <para>A index memory.</para>
77     /// <para></para>
78     /// </param>
79     /// <param name="memoryReservationStep">
80     /// <para>A memory reservation step.</para>
81     /// <para></para>
82     /// </param>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
85         ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
86         ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
87         ↪ IndexTreeType.Default, useLinkedList: true) { }
88
89     /// <summary>
90     /// <para>
91     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="dataMemory">

```

```

90    /// <para>A data memory.</para>
91    /// <para></para>
92    /// </param>
93    /// <param name="indexMemory">
94    /// <para>A index memory.</para>
95    /// <para></para>
96    /// </param>
97    /// <param name="memoryReservationStep">
98    /// <para>A memory reservation step.</para>
99    /// <para></para>
100   /// </param>
101   /// <param name="constants">
102   /// <para>A constants.</para>
103   /// <para></para>
104   /// </param>
105   [MethodImpl(MethodImplOptions.AggressiveInlining)]
106   public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
    ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
    ↪ IndexTreeType.Default, useLinkedList: true) { }

107   /// <summary>
108   /// <para>
109   /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
110   /// </para>
111   /// <para></para>
112   /// </summary>
113   /// <param name="dataMemory">
114   /// <para>A data memory.</para>
115   /// <para></para>
116   /// </param>
117   /// <param name="indexMemory">
118   /// <para>A index memory.</para>
119   /// <para></para>
120   /// </param>
121   /// <param name="memoryReservationStep">
122   /// <para>A memory reservation step.</para>
123   /// <para></para>
124   /// </param>
125   /// <param name="constants">
126   /// <para>A constants.</para>
127   /// <para></para>
128   /// </param>
129   /// <param name="indexTreeType">
130   /// <para>A index tree type.</para>
131   /// <para></para>
132   /// </param>
133   /// <param name="useLinkedList">
134   /// <para>A use linked list.</para>
135   /// <para></para>
136   /// </param>
137   [MethodImpl(MethodImplOptions.AggressiveInlining)]
138   public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
    ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↪ memoryReservationStep, constants, useLinkedList)
139   {
140   }
141   if (indexTreeType == IndexTreeType.SizeBalancedTree)
142   {
143       _createInternalSourceTreeMethods = () => new
    ↪ InternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
    ↪ _linksDataParts, _linksIndexParts, _header);
144       _createExternalSourceTreeMethods = () => new
    ↪ ExternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
    ↪ _linksDataParts, _linksIndexParts, _header);
145       _createInternalTargetTreeMethods = () => new
    ↪ InternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
    ↪ _linksDataParts, _linksIndexParts, _header);
146       _createExternalTargetTreeMethods = () => new
    ↪ ExternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
    ↪ _linksDataParts, _linksIndexParts, _header);
147   }
148   else
149   {
150       _createInternalSourceTreeMethods = () => new
    ↪ InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
    ↪ _linksDataParts, _linksIndexParts, _header);

```

```

151         _createExternalSourceTreeMethods = () => new
152             ↳ ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
153                 ↳ _linksDataParts, _linksIndexParts, _header);
154         _createInternalTargetTreeMethods = () => new
155             ↳ InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
156                 ↳ _linksDataParts, _linksIndexParts, _header);
157         _createExternalTargetTreeMethods = () => new
158             ↳ ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
159                 ↳ _linksDataParts, _linksIndexParts, _header);
160     }
161     Init(dataMemory, indexMemory);
162 }
163
164 /// <summary>
165 /// <para>
166 /// Sets the pointers using the specified data memory.
167 /// </para>
168 /// <para></para>
169 /// </summary>
170 /// <param name="dataMemory">
171 /// <para>The data memory.</para>
172 /// <para></para>
173 /// </param>
174 /// <param name="indexMemory">
175 /// <para>The index memory.</para>
176 /// <para></para>
177 /// </param>
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 protected override void SetPointers(IResizableDirectMemory dataMemory,
180     ↳ IResizableDirectMemory indexMemory)
181 {
182     _linksDataParts = (byte*)dataMemory.Pointer;
183     _linksIndexParts = (byte*)indexMemory.Pointer;
184     _header = _linksIndexParts;
185     if (_useLinkedList)
186     {
187         InternalSourcesListMethods = new
188             ↳ InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
189                 ↳ _linksIndexParts);
190     }
191     else
192     {
193         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
194     }
195     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
196     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
197     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
198     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
199 }
200
201 /// <summary>
202 /// <para>
203 /// Resets the pointers.
204 /// </para>
205 /// <para></para>
206 /// </summary>
207 [MethodImpl(MethodImplOptions.AggressiveInlining)]
208 protected override void ResetPointers()
209 {
210     base.ResetPointers();
211     _linksDataParts = null;
212     _linksIndexParts = null;
213     _header = null;
214 }
215
216 /// <summary>
217 /// <para>
218 /// Gets the header reference.
219 /// </para>
220 /// <para></para>
221 /// </summary>
222 /// <returns>
223 /// <para>A ref links header of t link</para>
224 /// <para></para>
225 /// </returns>
226 [MethodImpl(MethodImplOptions.AggressiveInlining)]
227 protected override ref LinksHeader<TLink> GetHeaderReference() => ref
228     ↳ AsRef<LinksHeader<TLink>>(_header);

```

```

219
220     /// <summary>
221     /// <para>
222     /// Gets the link data part reference using the specified link index.
223     /// </para>
224     /// <para></para>
225     /// </summary>
226     /// <param name="linkIndex">
227     /// <para>The link index.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>A ref raw link data part of t link</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
236     {
237         => ref AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (LinkDataPartSizeInBytes *
238         ConvertToInt64(linkIndex)));
239
240     /// <summary>
241     /// <para>
242     /// Gets the link index part reference using the specified link index.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="linkIndex">
247     /// <para>The link index.</para>
248     /// <para></para>
249     /// </param>
250     /// <returns>
251     /// <para>A ref raw link index part of t link</para>
252     /// <para></para>
253     /// </returns>
254     [MethodImpl(MethodImplOptions.AggressiveInlining)]
255     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
256     linkIndex) => ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
257     (LinkIndexPartSizeInBytes * ConvertToInt64(linkIndex)));
258 }
259 }

```

1.46 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Disposables;
5  using Platform.Singletons;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using Platform.Memory;
9  using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.Split.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the split memory links base.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <seealso cref="DisposableBase"/>
23     /// <seealso cref="ILinks{TLink}"/>
24     public abstract class SplitMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
25     {
26         private static readonly EqualityComparer<TLink> _equalityComparer =
27         => EqualityComparer<TLink>.Default;
28         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
29         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
30         => UncheckedConverter<TLink, long>.Default;
31         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
32         => UncheckedConverter<long, TLink>.Default;
33         private static readonly TLink _zero = default;
34         private static readonly TLink _one = Arithmetic.Increment(_zero);
35
36         /// <summary>Возвращает размер одной связи в байтах.</summary>
37         /// <remarks>

```



```

35  /// Используется только во вне класса, не рекомендуется использовать внутри.
36  /// Так как во вне не обязательно будет доступен unsafe C#.
37  /// </remarks>
38  public static readonly long LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;
39
40  /// <summary>
41  /// <para>
42  /// The size in bytes.
43  /// </para>
44  /// <para></para>
45  /// </summary>
46  public static readonly long LinkIndexPartSizeInBytes =
47  ↪ RawLinkIndexPart<TLink>.SizeInBytes;
48
49  /// <summary>
50  /// <para>
51  /// The size in bytes.
52  /// </para>
53  /// <para></para>
54  /// </summary>
55  public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
56
57  /// <summary>
58  /// <para>
59  /// The default links size step.
60  /// </para>
61  /// <para></para>
62  /// </summary>
63  public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
64
65  /// <summary>
66  /// <para>
67  /// The data memory.
68  /// </para>
69  /// <para></para>
70  /// </summary>
71  protected readonly IResizableDirectMemory _dataMemory;
72  /// <summary>
73  /// <para>
74  /// The index memory.
75  /// </para>
76  /// <para></para>
77  /// </summary>
78  protected readonly IResizableDirectMemory _indexMemory;
79  /// <summary>
80  /// <para>
81  /// The use linked list.
82  /// </para>
83  /// <para></para>
84  /// </summary>
85  protected readonly bool _useLinkedList;
86  /// <summary>
87  /// <para>
88  /// The data memory reservation step in bytes.
89  /// </para>
90  /// <para></para>
91  /// </summary>
92  protected readonly long _dataMemoryReservationStepInBytes;
93  /// <summary>
94  /// <para>
95  /// The index memory reservation step in bytes.
96  /// </para>
97  /// <para></para>
98  /// </summary>
99  protected readonly long _indexMemoryReservationStepInBytes;
100
101  /// <summary>
102  /// <para>
103  /// The internal sources list methods.
104  /// </para>
105  /// <para></para>
106  /// </summary>
107  protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
108  /// <summary>
109  /// <para>
110  /// The internal sources tree methods.
111  /// </para>
112  /// <para></para>
113  /// </summary>

```

```

113     protected ILinksTreeMethods<TLink> InternalSourcesTreeMethods;
114     /// <summary>
115     /// <para>
116     /// The external sources tree methods.
117     /// </para>
118     /// <para></para>
119     /// </summary>
120     protected ILinksTreeMethods<TLink> ExternalSourcesTreeMethods;
121     /// <summary>
122     /// <para>
123     /// The internal targets tree methods.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     protected ILinksTreeMethods<TLink> InternalTargetsTreeMethods;
128     /// <summary>
129     /// <para>
130     /// The external targets tree methods.
131     /// </para>
132     /// <para></para>
133     /// </summary>
134     protected ILinksTreeMethods<TLink> ExternalTargetsTreeMethods;
135     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
136     // → нужно использовать не список а дерево, так как так можно быстрее проверить на
137     // → наличие связи внутри
138     /// <summary>
139     /// <para>
140     /// The unused links list methods.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     protected ILinksListMethods<TLink> UnusedLinksListMethods;
145     /// <summary>
146     /// Возвращает общее число связей находящихся в хранилище.
147     /// </summary>
148     protected virtual TLink Total
149     {
150         [MethodImpl(MethodImplOptions.AggressiveInlining)]
151         get
152         {
153             ref var header = ref GetHeaderReference();
154             return Subtract(header.AllocatedLinks, header.FreeLinks);
155         }
156     }
157     /// <summary>
158     /// <para>
159     /// Gets the constants value.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     public virtual LinksConstants<TLink> Constants
164     {
165         [MethodImpl(MethodImplOptions.AggressiveInlining)]
166         get;
167     }
168     /// <summary>
169     /// <para>
170     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
171     /// </para>
172     /// <para></para>
173     /// </summary>
174     /// <param name="dataMemory">
175     /// <para>A data memory.</para>
176     /// <para></para>
177     /// </param>
178     /// <param name="indexMemory">
179     /// <para>A index memory.</para>
180     /// <para></para>
181     /// </param>
182     /// <param name="memoryReservationStep">
183     /// <para>A memory reservation step.</para>
184     /// <para></para>
185     /// </param>
186     /// <param name="constants">
187     /// <para>A constants.</para>
188     /// <para></para>
189     /// </param>

```

```

190     /// </param>
191     /// <param name="useLinkedList">
192     /// <para>A use linked list.</para>
193     /// <para></para>
194     /// </param>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants, bool
        ↪ useLinkedList)
197     {
198         _dataMemory = dataMemory;
199         _indexMemory = indexMemory;
200         _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
201         _indexMemoryReservationStepInBytes = memoryReservationStep *
        ↪ LinkIndexPartSizeInBytes;
202         _useLinkedList = useLinkedList;
203         Constants = constants;
204     }
205
206     /// <summary>
207     /// <para>
208     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
209     /// </para>
210     /// <para></para>
211     /// </summary>
212     /// <param name="dataMemory">
213     /// <para>A data memory.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="indexMemory">
217     /// <para>A index memory.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="memoryReservationStep">
221     /// <para>A memory reservation step.</para>
222     /// <para></para>
223     /// </param>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance, useLinkedList: true)
        ↪ { }
226
227     /// <summary>
228     /// <para>
229     /// Inits the data memory.
230     /// </para>
231     /// <para></para>
232     /// </summary>
233     /// <param name="dataMemory">
234     /// <para>The data memory.</para>
235     /// <para></para>
236     /// </param>
237     /// <param name="indexMemory">
238     /// <para>The index memory.</para>
239     /// <para></para>
240     /// </param>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory)
243     {
244         // Read allocated links from header
245         if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
246         {
247             indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
248         }
249         SetPointers(dataMemory, indexMemory);
250         ref var header = ref GetHeaderReference();
251         var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
252         // Adjust reserved capacity
253         var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254         if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255         {
256             minimumDataReservedCapacity = dataMemory.UsedCapacity;
257         }
258         if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259         {
260             minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;

```

```

261 }
262 var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263 if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264 {
265     minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266 }
267 if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268 {
269     minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270 }
271 // Check for alignment
272 if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273 {
274     minimumDataReservedCapacity = ((minimumDataReservedCapacity /
        ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
        ↪ _dataMemoryReservationStepInBytes;
275 }
276 if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
277 {
278     minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
        ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
        ↪ _indexMemoryReservationStepInBytes;
279 }
280 if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
281 {
282     dataMemory.ReservedCapacity = minimumDataReservedCapacity;
283 }
284 if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
285 {
286     indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
287 }
288 SetPointers(dataMemory, indexMemory);
289 header = ref GetHeaderReference();
290 // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
291 // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
292 dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
    ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
    ↪ zero link.
293 indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
    ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
294 // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
295 // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
296 header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
    ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
297 }
298
299 /// <summary>
300 /// <para>
301 /// Counts the substitution.
302 /// </para>
303 /// <para></para>
304 /// </summary>
305 /// <param name="restriction">
306 /// <para>The substitution.</para>
307 /// <para></para>
308 /// </param>
309 /// <exception cref="NotSupportedException">
310 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
311 /// <para></para>
312 /// </exception>
313 /// <returns>
314 /// <para>The link</para>
315 /// <para></para>
316 /// </returns>
317 [MethodImpl(MethodImplOptions.AggressiveInlining)]
318 public virtual TLink Count(IList<TLink>? restriction)
319 {
320     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
321     if (restriction.Count == 0)
322     {
323         return Total;
324     }
325     var constants = Constants;
326     var any = constants.Any;
327     var index = restriction[constants.IndexPart];
328     if (restriction.Count == 1)
329     {
330         if (AreEqual(index, any))

```

```

331     {
332         return Total;
333     }
334     return Exists(index) ? GetOne() : GetZero();
335 }
336 if (restriction.Count == 2)
337 {
338     var value = restriction[1];
339     if (AreEqual(index, any))
340     {
341         if (AreEqual(value, any))
342         {
343             return Total; // Any - как отсутствие ограничения
344         }
345         var externalReferencesRange = constants.ExternalReferencesRange;
346         if (externalReferencesRange.HasValue &&
347             ↪ externalReferencesRange.Value.Contains(value))
348         {
349             return Add(ExternalSourcesTreeMethods.CountUsages(value),
350                 ↪ ExternalTargetsTreeMethods.CountUsages(value));
351         }
352         else
353         {
354             if (_useLinkedList)
355             {
356                 return Add(InternalSourcesListMethods.CountUsages(value),
357                     ↪ InternalTargetsTreeMethods.CountUsages(value));
358             }
359             else
360             {
361                 return Add(InternalSourcesTreeMethods.CountUsages(value),
362                     ↪ InternalTargetsTreeMethods.CountUsages(value));
363             }
364         }
365     }
366     else
367     {
368         if (!Exists(index))
369         {
370             return GetZero();
371         }
372         if (AreEqual(value, any))
373         {
374             return GetOne();
375         }
376         ref var storedLinkValue = ref GetLinkDataPartReference(index);
377         if (AreEqual(storedLinkValue.Source, value) ||
378             ↪ AreEqual(storedLinkValue.Target, value))
379         {
380             return GetOne();
381         }
382         return GetZero();
383     }
384 }
385 if (restriction.Count == 3)
386 {
387     var externalReferencesRange = constants.ExternalReferencesRange;
388     var source = restriction[constants.SourcePart];
389     var target = restriction[constants.TargetPart];
390     if (AreEqual(index, any))
391     {
392         if (AreEqual(source, any) && AreEqual(target, any))
393         {
394             return Total;
395         }
396         else if (AreEqual(source, any))
397         {
398             if (externalReferencesRange.HasValue &&
399                 ↪ externalReferencesRange.Value.Contains(target))
400             {
401                 return ExternalTargetsTreeMethods.CountUsages(target);
402             }
403             else
404             {
405                 return InternalTargetsTreeMethods.CountUsages(target);
406             }
407         }
408         else if (AreEqual(target, any))

```

```

403 {
404     if (externalReferencesRange.HasValue &&
        ↪ externalReferencesRange.Value.Contains(source))
405     {
406         return ExternalSourcesTreeMethods.CountUsages(source);
407     }
408     else
409     {
410         if (_useLinkedList)
411         {
412             return InternalSourcesListMethods.CountUsages(source);
413         }
414         else
415         {
416             return InternalSourcesTreeMethods.CountUsages(source);
417         }
418     }
419 }
420 else //if(source != Any && target != Any)
421 {
422     // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
423     TLink link;
424     if (externalReferencesRange.HasValue)
425     {
426         if (externalReferencesRange.Value.Contains(source) &&
            ↪ externalReferencesRange.Value.Contains(target))
427         {
428             link = ExternalSourcesTreeMethods.Search(source, target);
429         }
430         else if (externalReferencesRange.Value.Contains(source))
431         {
432             link = InternalTargetsTreeMethods.Search(source, target);
433         }
434         else if (externalReferencesRange.Value.Contains(target))
435         {
436             if (_useLinkedList)
437             {
438                 link = ExternalSourcesTreeMethods.Search(source, target);
439             }
440             else
441             {
442                 link = InternalSourcesTreeMethods.Search(source, target);
443             }
444         }
445         else
446         {
447             if (_useLinkedList ||
                ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                ↪ InternalTargetsTreeMethods.CountUsages(target)))
448             {
449                 link = InternalTargetsTreeMethods.Search(source, target);
450             }
451             else
452             {
453                 link = InternalSourcesTreeMethods.Search(source, target);
454             }
455         }
456     }
457     else
458     {
459         if (_useLinkedList ||
            ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
            ↪ InternalTargetsTreeMethods.CountUsages(target)))
460         {
461             link = InternalTargetsTreeMethods.Search(source, target);
462         }
463         else
464         {
465             link = InternalSourcesTreeMethods.Search(source, target);
466         }
467     }
468     return AreEqual(link, constants.Null) ? GetZero() : GetOne();
469 }
470 }
471 else
472 {
473     if (!Exists(index))
474     {

```

```

475         return GetZero();
476     }
477     if (AreEqual(source, any) && AreEqual(target, any))
478     {
479         return GetOne();
480     }
481     ref var storedLinkValue = ref GetLinkDataPartReference(index);
482     if (!AreEqual(source, any) && !AreEqual(target, any))
483     {
484         if (AreEqual(storedLinkValue.Source, source) &&
485             ↪ AreEqual(storedLinkValue.Target, target))
486         {
487             return GetOne();
488         }
489         return GetZero();
490     }
491     var value = default(TLink);
492     if (AreEqual(source, any))
493     {
494         value = target;
495     }
496     if (AreEqual(target, any))
497     {
498         value = source;
499     }
500     if (AreEqual(storedLinkValue.Source, value) ||
501         ↪ AreEqual(storedLinkValue.Target, value))
502     {
503         return GetOne();
504     }
505     return GetZero();
506 }
507 }
508 throw new NotSupportedException("Другие размеры и способы ограничений не
509 ↪ поддерживаются.");
510 }
511
512 /// <summary>
513 /// <para>
514 /// Eaches the handler.
515 /// </para>
516 /// <para></para>
517 /// </summary>
518 /// <param name="handler">
519 /// <para>The handler.</para>
520 /// <para></para>
521 /// </param>
522 /// <param name="restriction">
523 /// <para>The substitution.</para>
524 /// <para></para>
525 /// </param>
526 /// <exception cref="NotSupportedException">
527 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
528 /// <para></para>
529 /// </exception>
530 /// <returns>
531 /// <para>The link</para>
532 /// <para></para>
533 /// </returns>
534 [MethodImpl(MethodImplOptions.AggressiveInlining)]
535 public virtual TLink Each(ICollection<TLink> restriction, ReadHandler<TLink>? handler)
536 {
537     var constants = Constants;
538     var @break = constants.Break;
539     if (restriction.Count == 0)
540     {
541         for (var link = GetOne(); LessOrEqualThan(link,
542             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
543         {
544             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
545             {
546                 return @break;
547             }
548         }
549         return @break;
550     }
551     var @continue = constants.Continue;
552     var any = constants.Any;

```

```

549 var index = restriction[constants.IndexPart];
550 if (restriction.Count == 1)
551 {
552     if (AreEqual(index, any))
553     {
554         return Each(Array.Empty<TLink>(), handler);
555     }
556     if (!Exists(index))
557     {
558         return @continue;
559     }
560     return handler(GetLinkStruct(index));
561 }
562 if (restriction.Count == 2)
563 {
564     var value = restriction[1];
565     if (AreEqual(index, any))
566     {
567         if (AreEqual(value, any))
568         {
569             return Each(Array.Empty<TLink>(), handler);
570         }
571         if (AreEqual(Each(new Link<TLink>(index, value, any), handler), @break))
572         {
573             return @break;
574         }
575         return Each(new Link<TLink>(index, any, value), handler);
576     }
577     else
578     {
579         if (!Exists(index))
580         {
581             return @continue;
582         }
583         if (AreEqual(value, any))
584         {
585             return handler(GetLinkStruct(index));
586         }
587         ref var storedLinkValue = ref GetLinkDataPartReference(index);
588         if (AreEqual(storedLinkValue.Source, value) ||
589             AreEqual(storedLinkValue.Target, value))
590         {
591             return handler(GetLinkStruct(index));
592         }
593         return @continue;
594     }
595 }
596 if (restriction.Count == 3)
597 {
598     var externalReferencesRange = constants.ExternalReferencesRange;
599     var source = restriction[constants.SourcePart];
600     var target = restriction[constants.TargetPart];
601     if (AreEqual(index, any))
602     {
603         if (AreEqual(source, any) && AreEqual(target, any))
604         {
605             return Each(Array.Empty<TLink>(), handler);
606         }
607         else if (AreEqual(source, any))
608         {
609             if (externalReferencesRange.HasValue &&
610                 ⇨ externalReferencesRange.Value.Contains(target))
611             {
612                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
613             }
614             else
615             {
616                 return InternalTargetsTreeMethods.EachUsage(target, handler);
617             }
618         }
619         else if (AreEqual(target, any))
620         {
621             if (externalReferencesRange.HasValue &&
622                 ⇨ externalReferencesRange.Value.Contains(source))
623             {
624                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
625             }
626             else

```



```

625     {
626         if (_useLinkedList)
627         {
628             return InternalSourcesListMethods.EachUsage(source, handler);
629         }
630         else
631         {
632             return InternalSourcesTreeMethods.EachUsage(source, handler);
633         }
634     }
635 }
636 else //if(source != Any && target != Any)
637 {
638     TLink link;
639     if (externalReferencesRange.HasValue)
640     {
641         if (externalReferencesRange.Value.Contains(source) &&
642             ↪ externalReferencesRange.Value.Contains(target))
643         {
644             link = ExternalSourcesTreeMethods.Search(source, target);
645         }
646         else if (externalReferencesRange.Value.Contains(source))
647         {
648             link = InternalTargetsTreeMethods.Search(source, target);
649         }
650         else if (externalReferencesRange.Value.Contains(target))
651         {
652             if (_useLinkedList)
653             {
654                 link = ExternalSourcesTreeMethods.Search(source, target);
655             }
656             else
657             {
658                 link = InternalSourcesTreeMethods.Search(source, target);
659             }
660         }
661         else
662         {
663             if (_useLinkedList ||
664                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
665                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
666             {
667                 link = InternalTargetsTreeMethods.Search(source, target);
668             }
669             else
670             {
671                 link = InternalSourcesTreeMethods.Search(source, target);
672             }
673         }
674     }
675     }
676     }
677     else
678     {
679         if (_useLinkedList ||
680             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
681             ↪ InternalTargetsTreeMethods.CountUsages(target)))
682         {
683             link = InternalTargetsTreeMethods.Search(source, target);
684         }
685         else
686         {
687             link = InternalSourcesTreeMethods.Search(source, target);
688         }
689     }
690     return AreEqual(link, constants.Null) ? @continue :
691         ↪ handler(GetLinkStruct(link));
692 }
693 }
694 else
695 {
696     if (!Exists(index))
697     {
698         return @continue;
699     }
700     if (AreEqual(source, any) && AreEqual(target, any))
701     {
702         return handler(GetLinkStruct(index));
703     }
704     ref var storedLinkValue = ref GetLinkDataPartReference(index);

```

```

697         if (!AreEqual(source, any) && !AreEqual(target, any))
698         {
699             if (AreEqual(storedLinkValue.Source, source) &&
700                 AreEqual(storedLinkValue.Target, target))
701             {
702                 return handler(GetLinkStruct(index));
703             }
704             return @continue;
705         }
706         var value = default(TLink);
707         if (AreEqual(source, any))
708         {
709             value = target;
710         }
711         if (AreEqual(target, any))
712         {
713             value = source;
714         }
715         if (AreEqual(storedLinkValue.Source, value) ||
716             AreEqual(storedLinkValue.Target, value))
717         {
718             return handler(GetLinkStruct(index));
719         }
720         return @continue;
721     }
722 }
723 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
724 }
725
726 /// <remarks>
727 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
    ↳ в другом месте (но не в менеджере памяти, а в логике Links)
728 /// </remarks>
729 [MethodImpl(MethodImplOptions.AggressiveInlining)]
730 public virtual TLink Update(ICollection<TLink> restriction, ICollection<TLink> substitution,
    ↳ WriteHandler<TLink> handler)
731 {
732     var constants = Constants;
733     var @null = constants.Null;
734     var externalReferencesRange = constants.ExternalReferencesRange;
735     var linkIndex = restriction[constants.IndexPart];
736     var before = GetLinkStruct(linkIndex);
737     ref var link = ref GetLinkDataPartReference(linkIndex);
738     var source = link.Source;
739     var target = link.Target;
740     ref var header = ref GetHeaderReference();
741     ref var rootAsSource = ref header.RootAsSource;
742     ref var rootAsTarget = ref header.RootAsTarget;
743     // Будет корректно работать только в том случае, если пространство выделенной связи
    ↳ предварительно заполнено нулями
744     if (!AreEqual(source, @null))
745     {
746         if (externalReferencesRange.HasValue &&
    ↳ externalReferencesRange.Value.Contains(source))
747         {
748             ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
749         }
750         else
751         {
752             if (_useLinkedList)
753             {
754                 InternalSourcesListMethods.Detach(source, linkIndex);
755             }
756             else
757             {
758                 InternalSourcesTreeMethods.Detach(ref
    ↳ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
759             }
760         }
761     }
762     if (!AreEqual(target, @null))
763     {
764         if (externalReferencesRange.HasValue &&
    ↳ externalReferencesRange.Value.Contains(target))
765         {
766             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
767         }

```

```

768     else
769     {
770         InternalTargetsTreeMethods.Detach(ref
            ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
771     }
772 }
773 source = link.Source = substitution[constants.SourcePart];
774 target = link.Target = substitution[constants.TargetPart];
775 if (!AreEqual(source, @null))
776 {
777     if (externalReferencesRange.HasValue &&
            ↪ externalReferencesRange.Value.Contains(source))
778     {
779         ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
780     }
781     else
782     {
783         if (_useLinkedList)
784         {
785             InternalSourcesListMethods.AttachAsLast(source, linkIndex);
786         }
787         else
788         {
789             InternalSourcesTreeMethods.Attach(ref
                ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
790         }
791     }
792 }
793 if (!AreEqual(target, @null))
794 {
795     if (externalReferencesRange.HasValue &&
            ↪ externalReferencesRange.Value.Contains(target))
796     {
797         ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
798     }
799     else
800     {
801         InternalTargetsTreeMethods.Attach(ref
            ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
802     }
803 }
804 return handler != null ? handler(before, new Link<TLink>(linkIndex, source, target))
            ↪ : Constants.Continue;
805 }
806
807 /// <remarks>
808 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
            ↪ пространство
809 /// </remarks>
810 [MethodImpl(MethodImplOptions.AggressiveInlining)]
811 public virtual TLink Create(ICollection<TLink> substitution, WriteHandler<TLink> handler)
812 {
813     ref var header = ref GetHeaderReference();
814     var freeLink = header.FirstFreeLink;
815     if (!AreEqual(freeLink, Constants.Null))
816     {
817         UnusedLinksListMethods.Detach(freeLink);
818     }
819     else
820     {
821         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
822         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
823         {
824             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
825         }
826         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
827         {
828             _dataMemory.ReservedCapacity += _dataMemory.ReservationStepInBytes;
829             _indexMemory.ReservedCapacity += _indexMemory.ReservationStepInBytes;
830             SetPointers(_dataMemory, _indexMemory);
831             header = ref GetHeaderReference();
832             header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
                ↪ LinkDataPartSizeInBytes);
833         }
834         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
835         _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
836         _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
837     }

```

```

838     return handler != null ? handler(null, GetLinkStruct(freeLink)) : Constants.Continue;
839 }
840
841 /// <summary>
842 /// <para>
843 /// Deletes the substitution.
844 /// </para>
845 /// <para></para>
846 /// </summary>
847 /// <param name="restriction">
848 /// <para>The substitution.</para>
849 /// <para></para>
850 /// </param>
851 [MethodImpl(MethodImplOptions.AggressiveInlining)]
852 public virtual TLink Delete(ICollection<TLink> restriction, WriteHandler<TLink> handler)
853 {
854     ref var header = ref GetHeaderReference();
855     var link = restriction[Constants.IndexPart];
856     var before = GetLinkStruct(link);
857     if (LessThan(link, header.AllocatedLinks))
858     {
859         UnusedLinksListMethods.AttachAsFirst(link);
860     }
861     else if (AreEqual(link, header.AllocatedLinks))
862     {
863         header.AllocatedLinks = Decrement(header.AllocatedLinks);
864         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
865         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
866         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
867         // → пока не дойдём до первой существующей связи
868         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
869         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
870             → IsUnusedLink(header.AllocatedLinks))
871         {
872             UnusedLinksListMethods.Detach(header.AllocatedLinks);
873             header.AllocatedLinks = Decrement(header.AllocatedLinks);
874             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
875             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
876         }
877     }
878     return handler != null ? handler(before, null) : Constants.Continue;
879 }
880
881 /// <summary>
882 /// <para>
883 /// Gets the link struct using the specified link index.
884 /// </para>
885 /// <para></para>
886 /// </summary>
887 /// <param name="linkIndex">
888 /// <para>The link index.</para>
889 /// <para></para>
890 /// </param>
891 /// <returns>
892 /// <para>A list of t link</para>
893 /// <para></para>
894 /// </returns>
895 [MethodImpl(MethodImplOptions.AggressiveInlining)]
896 public ICollection<TLink> GetLinkStruct(TLink linkIndex)
897 {
898     ref var link = ref GetLinkDataPartReference(linkIndex);
899     return new Link<TLink>(linkIndex, link.Source, link.Target);
900 }
901
902 /// <remarks>
903 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
904 → адрес реально поменялся
905 ///
906 /// Указатель this.links может быть в том же месте,
907 /// так как 0-я связь не используется и имеет такой же размер как Header,
908 /// поэтому header размещается в том же месте, что и 0-я связь
909 /// </remarks>
910 [MethodImpl(MethodImplOptions.AggressiveInlining)]
911 protected abstract void SetPointers(ICollection<TLink> dataMemory,
912     → ICollection<TLink> indexMemory);
913
914 /// <summary>
915 /// <para>

```

```

912    /// Resets the pointers.
913    /// </para>
914    /// <para></para>
915    /// </summary>
916    [MethodImpl(MethodImplOptions.AggressiveInlining)]
917    protected virtual void ResetPointers()
918    {
919        InternalSourcesListMethods = null;
920        InternalSourcesTreeMethods = null;
921        ExternalSourcesTreeMethods = null;
922        InternalTargetsTreeMethods = null;
923        ExternalTargetsTreeMethods = null;
924        UnusedLinksListMethods = null;
925    }
926
927    /// <summary>
928    /// <para>
929    /// Gets the header reference.
930    /// </para>
931    /// <para></para>
932    /// </summary>
933    /// <returns>
934    /// <para>A ref links header of t link</para>
935    /// <para></para>
936    /// </returns>
937    [MethodImpl(MethodImplOptions.AggressiveInlining)]
938    protected abstract ref LinksHeader<TLink> GetHeaderReference();
939
940    /// <summary>
941    /// <para>
942    /// Gets the link data part reference using the specified link index.
943    /// </para>
944    /// <para></para>
945    /// </summary>
946    /// <param name="linkIndex">
947    /// <para>The link index.</para>
948    /// <para></para>
949    /// </param>
950    /// <returns>
951    /// <para>A ref raw link data part of t link</para>
952    /// <para></para>
953    /// </returns>
954    [MethodImpl(MethodImplOptions.AggressiveInlining)]
955    protected abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex);
956
957    /// <summary>
958    /// <para>
959    /// Gets the link index part reference using the specified link index.
960    /// </para>
961    /// <para></para>
962    /// </summary>
963    /// <param name="linkIndex">
964    /// <para>The link index.</para>
965    /// <para></para>
966    /// </param>
967    /// <returns>
968    /// <para>A ref raw link index part of t link</para>
969    /// <para></para>
970    /// </returns>
971    [MethodImpl(MethodImplOptions.AggressiveInlining)]
972    protected abstract ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
    ↪ linkIndex);
973
974    /// <summary>
975    /// <para>
976    /// Determines whether this instance exists.
977    /// </para>
978    /// <para></para>
979    /// </summary>
980    /// <param name="link">
981    /// <para>The link.</para>
982    /// <para></para>
983    /// </param>
984    /// <returns>
985    /// <para>The bool</para>
986    /// <para></para>
987    /// </returns>
988    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

989     protected virtual bool Exists(TLink link)
990     => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
991         && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
992         && !IsUnusedLink(link);
993
994     /// <summary>
995     /// <para>
996     /// Determines whether this instance is unused link.
997     /// </para>
998     /// <para></para>
999     /// </summary>
1000     /// <param name="linkIndex">
1001     /// <para>The link index.</para>
1002     /// <para></para>
1003     /// </param>
1004     /// <returns>
1005     /// <para>The bool</para>
1006     /// <para></para>
1007     /// </returns>
1008     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1009     protected virtual bool IsUnusedLink(TLink linkIndex)
1010     {
1011         if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
1012             ↳ is not needed
1013         {
1014             // TODO: Reduce access to memory in different location (should be enough to use
1015             ↳ just linkIndexPart)
1016             ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
1017             ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
1018             return AreEqual(linkIndexPart.SizeAsTarget, default) &&
1019                 ↳ !AreEqual(linkDataPart.Source, default);
1020         }
1021         else
1022         {
1023             return true;
1024         }
1025     }
1026
1027     /// <summary>
1028     /// <para>
1029     /// Gets the one.
1030     /// </para>
1031     /// <para></para>
1032     /// </summary>
1033     /// <returns>
1034     /// <para>The link</para>
1035     /// <para></para>
1036     /// </returns>
1037     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1038     protected virtual TLink GetOne() => _one;
1039
1040     /// <summary>
1041     /// <para>
1042     /// Gets the zero.
1043     /// </para>
1044     /// <para></para>
1045     /// </summary>
1046     /// <returns>
1047     /// <para>The link</para>
1048     /// <para></para>
1049     /// </returns>
1050     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1051     protected virtual TLink GetZero() => default;
1052
1053     /// <summary>
1054     /// <para>
1055     /// Determines whether this instance are equal.
1056     /// </para>
1057     /// <para></para>
1058     /// </summary>
1059     /// <param name="first">
1060     /// <para>The first.</para>
1061     /// <para></para>
1062     /// </param>
1063     /// <param name="second">
1064     /// <para>The second.</para>
1065     /// <para></para>
1066     /// </param>

```

```

1064    /// <returns>
1065    /// <para>The bool</para>
1066    /// <para></para>
1067    /// </returns>
1068    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1069    protected virtual bool AreEqual(TLink first, TLink second) =>
1070        ↪ _equalityComparer.Equals(first, second);
1071
1072    /// <summary>
1073    /// <para>
1074    /// Determines whether this instance less than.
1075    /// </para>
1076    /// <para></para>
1077    /// </summary>
1078    /// <param name="first">
1079    /// <para>The first.</para>
1080    /// <para></para>
1081    /// </param>
1082    /// <param name="second">
1083    /// <para>The second.</para>
1084    /// <para></para>
1085    /// </param>
1086    /// <returns>
1087    /// <para>The bool</para>
1088    /// <para></para>
1089    /// </returns>
1090    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1091    protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
1092        ↪ second) < 0;
1093
1094    /// <summary>
1095    /// <para>
1096    /// Determines whether this instance less or equal than.
1097    /// </para>
1098    /// <para></para>
1099    /// </summary>
1100    /// <param name="first">
1101    /// <para>The first.</para>
1102    /// <para></para>
1103    /// </param>
1104    /// <param name="second">
1105    /// <para>The second.</para>
1106    /// <para></para>
1107    /// </param>
1108    /// <returns>
1109    /// <para>The bool</para>
1110    /// <para></para>
1111    /// </returns>
1112    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1113    protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
1114        ↪ _comparer.Compare(first, second) <= 0;
1115
1116    /// <summary>
1117    /// <para>
1118    /// Determines whether this instance greater than.
1119    /// </para>
1120    /// <para></para>
1121    /// </summary>
1122    /// <param name="first">
1123    /// <para>The first.</para>
1124    /// <para></para>
1125    /// </param>
1126    /// <param name="second">
1127    /// <para>The second.</para>
1128    /// <para></para>
1129    /// </param>
1130    /// <returns>
1131    /// <para>The bool</para>
1132    /// <para></para>
1133    /// </returns>
1134    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1135    protected virtual bool GreaterThan(TLink first, TLink second) =>
1136        ↪ _comparer.Compare(first, second) > 0;
1137
1138    /// <summary>
1139    /// <para>
1140    /// Determines whether this instance greater or equal than.
1141    /// </para>

```

```

1138     /// <para></para>
1139     /// </summary>
1140     /// <param name="first">
1141     /// <para>The first.</para>
1142     /// <para></para>
1143     /// </param>
1144     /// <param name="second">
1145     /// <para>The second.</para>
1146     /// <para></para>
1147     /// </param>
1148     /// <returns>
1149     /// <para>The bool</para>
1150     /// <para></para>
1151     /// </returns>
1152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1153     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
1154         ↪ _comparer.Compare(first, second) >= 0;
1155
1156     /// <summary>
1157     /// <para>
1158     /// Converts the to int 64 using the specified value.
1159     /// </para>
1160     /// <para></para>
1161     /// </summary>
1162     /// <param name="value">
1163     /// <para>The value.</para>
1164     /// <para></para>
1165     /// </param>
1166     /// <returns>
1167     /// <para>The long</para>
1168     /// <para></para>
1169     /// </returns>
1170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1171     protected virtual long ConvertToInt64(TLink value) =>
1172         ↪ _addressToInt64Converter.Convert(value);
1173
1174     /// <summary>
1175     /// <para>
1176     /// Converts the to address using the specified value.
1177     /// </para>
1178     /// <para></para>
1179     /// </summary>
1180     /// <param name="value">
1181     /// <para>The value.</para>
1182     /// <para></para>
1183     /// </param>
1184     /// <returns>
1185     /// <para>The link</para>
1186     /// <para></para>
1187     /// </returns>
1188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1189     protected virtual TLink ConvertToAddress(long value) =>
1190         ↪ _int64ToAddressConverter.Convert(value);
1191
1192     /// <summary>
1193     /// <para>
1194     /// Adds the first.
1195     /// </para>
1196     /// <para></para>
1197     /// </summary>
1198     /// <param name="first">
1199     /// <para>The first.</para>
1200     /// <para></para>
1201     /// </param>
1202     /// <param name="second">
1203     /// <para>The second.</para>
1204     /// <para></para>
1205     /// </param>
1206     /// <returns>
1207     /// <para>The link</para>
1208     /// <para></para>
1209     /// </returns>
1210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1211     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
        ↪ second);

```



```

1212    /// Subtracts the first.
1213    /// </para>
1214    /// <para></para>
1215    /// </summary>
1216    /// <param name="first">
1217    /// <para>The first.</para>
1218    /// <para></para>
1219    /// </param>
1220    /// <param name="second">
1221    /// <para>The second.</para>
1222    /// <para></para>
1223    /// </param>
1224    /// <returns>
1225    /// <para>The link</para>
1226    /// <para></para>
1227    /// </returns>
1228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1229    protected virtual TLink Subtract(TLink first, TLink second) =>
        ↪ Arithmetic<TLink>.Subtract(first, second);
1230
1231    /// <summary>
1232    /// <para>
1233    /// Increments the link.
1234    /// </para>
1235    /// <para></para>
1236    /// </summary>
1237    /// <param name="link">
1238    /// <para>The link.</para>
1239    /// <para></para>
1240    /// </param>
1241    /// <returns>
1242    /// <para>The link</para>
1243    /// <para></para>
1244    /// </returns>
1245    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1246    protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
1247
1248    /// <summary>
1249    /// <para>
1250    /// Decrements the link.
1251    /// </para>
1252    /// <para></para>
1253    /// </summary>
1254    /// <param name="link">
1255    /// <para>The link.</para>
1256    /// <para></para>
1257    /// </param>
1258    /// <returns>
1259    /// <para>The link</para>
1260    /// <para></para>
1261    /// </returns>
1262    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1263    protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
1264
1265    #region Disposable
1266
1267    /// <summary>
1268    /// <para>
1269    /// Gets the allow multiple dispose calls value.
1270    /// </para>
1271    /// <para></para>
1272    /// </summary>
1273    protected override bool AllowMultipleDisposeCalls
1274    {
1275        [MethodImpl(MethodImplOptions.AggressiveInlining)]
1276        get => true;
1277    }
1278
1279    /// <summary>
1280    /// <para>
1281    /// Disposes the manual.
1282    /// </para>
1283    /// <para></para>
1284    /// </summary>
1285    /// <param name="manual">
1286    /// <para>The manual.</para>
1287    /// <para></para>
1288    /// </param>

```

```

1289     /// <param name="wasDisposed">
1290     /// <para>The was disposed.</para>
1291     /// <para></para>
1292     /// </param>
1293     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1294     protected override void Dispose(bool manual, bool wasDisposed)
1295     {
1296         if (!wasDisposed)
1297         {
1298             ResetPointers();
1299             _dataMemory.DisposeIfPossible();
1300             _indexMemory.DisposeIfPossible();
1301         }
1302     }
1303
1304     #endregion
1305 }
1306 }

```

1.47 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split.Generic
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unused links list methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLink}"/>
17     /// <seealso cref="ILinksListMethods{TLink}"/>
18     public unsafe class UnusedLinksListMethods<TLink> :
19     ↪ AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
20     {
21         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
22         ↪ UncheckedConverter<TLink, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UnusedLinksListMethods(byte* links, byte* header)
42         {
43             _links = links;
44             _header = header;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Gets the header reference.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <returns>
54         /// <para>A ref links header of t link</para>
55         /// <para></para>
56         /// </returns>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
59         ↪ AsRef<LinksHeader<TLink>>(_header);

```

```

57
58     /// <summary>
59     /// <para>
60     /// Gets the link data part reference using the specified link.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="link">
65     /// <para>The link.</para>
66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>A ref raw link data part of t link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
74     ↪ AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
75     ↪ _addressToInt64Converter.Convert(link)));
76
77     /// <summary>
78     /// <para>
79     /// Gets the first.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <returns>
84     /// <para>The link</para>
85     /// <para></para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
89
90     /// <summary>
91     /// <para>
92     /// Gets the last.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     /// <returns>
97     /// <para>The link</para>
98     /// <para></para>
99     /// </returns>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
102
103    /// <summary>
104    /// <para>
105    /// Gets the previous using the specified element.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    /// <param name="element">
110    /// <para>The element.</para>
111    /// <para></para>
112    /// </param>
113    /// <returns>
114    /// <para>The link</para>
115    /// <para></para>
116    /// </returns>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    protected override TLink GetPrevious(TLink element) =>
119    ↪ GetLinkDataPartReference(element).Source;
120
121    /// <summary>
122    /// <para>
123    /// Gets the next using the specified element.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="element">
128    /// <para>The element.</para>
129    /// <para></para>
130    /// </param>
131    /// <returns>
132    /// <para>The link</para>
133    /// <para></para>
134    /// </returns>

```

```

132 [MethodImpl(MethodImplOptions.AggressiveInlining)]
133 protected override TLink GetNext(TLink element) =>
    ↳ GetLinkDataPartReference(element).Target;
134
135 /// <summary>
136 /// <para>
137 /// Gets the size.
138 /// </para>
139 /// <para></para>
140 /// </summary>
141 /// <returns>
142 /// <para>The link</para>
143 /// <para></para>
144 /// </returns>
145 [MethodImpl(MethodImplOptions.AggressiveInlining)]
146 protected override TLink GetSize() => GetHeaderReference().FreeLinks;
147
148 /// <summary>
149 /// <para>
150 /// Sets the first using the specified element.
151 /// </para>
152 /// <para></para>
153 /// </summary>
154 /// <param name="element">
155 /// <para>The element.</para>
156 /// <para></para>
157 /// </param>
158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
    ↳ element;
160
161 /// <summary>
162 /// <para>
163 /// Sets the last using the specified element.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 /// <param name="element">
168 /// <para>The element.</para>
169 /// <para></para>
170 /// </param>
171 [MethodImpl(MethodImplOptions.AggressiveInlining)]
172 protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
    ↳ element;
173
174 /// <summary>
175 /// <para>
176 /// Sets the previous using the specified element.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <param name="element">
181 /// <para>The element.</para>
182 /// <para></para>
183 /// </param>
184 /// <param name="previous">
185 /// <para>The previous.</para>
186 /// <para></para>
187 /// </param>
188 [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 protected override void SetPrevious(TLink element, TLink previous) =>
    ↳ GetLinkDataPartReference(element).Source = previous;
190
191 /// <summary>
192 /// <para>
193 /// Sets the next using the specified element.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="element">
198 /// <para>The element.</para>
199 /// <para></para>
200 /// </param>
201 /// <param name="next">
202 /// <para>The next.</para>
203 /// <para></para>
204 /// </param>
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

206     protected override void SetNext(TLink element, TLink next) =>
207         ↪ GetLinkDataPartReference(element).Target = next;
208
209     /// <summary>
210     /// <para>
211     /// Sets the size using the specified size.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="size">
216     /// <para>The size.</para>
217     /// <para></para>
218     /// </param>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
221 }

```

1.48 ./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link data part.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLinkDataPart<TLink> : IEquatable<RawLinkDataPart<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkDataPart<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink Source;
36
37         /// <summary>
38         /// <para>
39         /// The target.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink Target;
44
45         /// <summary>
46         /// <para>
47         /// Determines whether this instance equals.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="obj">
52         /// <para>The obj.</para>
53         /// <para></para>
54         /// </param>
55         /// <returns>
56         /// <para>The bool</para>
57         /// <para></para>
58         /// </returns>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public override bool Equals(object obj) => obj is RawLinkDataPart<TLink> link ?
61             ↪ Equals(link) : false;

```

```

59
60     /// <summary>
61     /// <para>
62     /// Determines whether this instance equals.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="other">
67     /// <para>The other.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The bool</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public bool Equals(RawLinkDataPart<TLink> other)
76         => _equalityComparer.Equals(Source, other.Source)
77         && _equalityComparer.Equals(Target, other.Target);
78
79     /// <summary>
80     /// <para>
81     /// Gets the hash code.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <returns>
86     /// <para>The int</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public override int GetHashCode() => (Source, Target).GetHashCode();
91
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public static bool operator ==(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
94         ↪ right) => left.Equals(right);
95
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public static bool operator !=(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
98         ↪ right) => !(left == right);
99 }
100 }

```

1.49 ./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link index part.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLinkIndexPart<TLink> : IEquatable<RawLinkIndexPart<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The root as source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink RootAsSource;

```

```

35     /// <summary>
36     /// <para>
37     /// The left as source.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLink LeftAsSource;
42     /// <summary>
43     /// <para>
44     /// The right as source.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public TLink RightAsSource;
49     /// <summary>
50     /// <para>
51     /// The size as source.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLink SizeAsSource;
56     /// <summary>
57     /// <para>
58     /// The root as target.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLink RootAsTarget;
63     /// <summary>
64     /// <para>
65     /// The left as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLink LeftAsTarget;
70     /// <summary>
71     /// <para>
72     /// The right as target.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLink RightAsTarget;
77     /// <summary>
78     /// <para>
79     /// The size as target.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLink SizeAsTarget;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100     public override bool Equals(object obj) => obj is RawLinkIndexPart<TLink> link ?
        ↳ Equals(link) : false;
101
102     /// <summary>
103     /// <para>
104     /// Determines whether this instance equals.
105     /// </para>
106     /// <para></para>
107     /// </summary>
108     /// <param name="other">
109     /// <para>The other.</para>
110     /// <para></para>
111     /// </param>

```

```

112     /// <returns>
113     /// <para>The bool</para>
114     /// <para></para>
115     /// </returns>
116     [MethodImpl(MethodImplOptions.AggressiveInlining)]
117     public bool Equals(RawLinkIndexPart<TLink> other)
118     => _equalityComparer.Equals(RootAsSource, other.RootAsSource)
119     && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
120     && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
121     && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
122     && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
123     && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
124     && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
125     && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
126
127     /// <summary>
128     /// <para>
129     /// Gets the hash code.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <returns>
134     /// <para>The int</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public override int GetHashCode() => (RootAsSource, LeftAsSource, RightAsSource,
139     ↪ SizeAsSource, RootAsTarget, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
143     ↪ right) => left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
147     ↪ right) => !(left == right);
148 }
149 }

```

1.50 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTree

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 external links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16    /// <seealso cref="ILinksTreeMethods{TLink}"/>
17    public unsafe abstract class UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
18    ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
19    {
20        /// <summary>
21        /// <para>
22        /// The links data parts.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
27
28        /// <summary>
29        /// <para>
30        /// The links index parts.
31        /// </para>
32        /// <para></para>
33        /// </summary>
34        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
35
36        /// <summary>
37        /// <para>
38        /// The header.
39        /// </para>
40        /// <para></para>
41        /// </summary>
42    }
43 }

```



```

38     /// </summary>
39     protected new readonly LinksHeader<TLink>* Header;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see
44     ↪ cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="constants">
49     /// <para>A constants.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="linksDataParts">
53     /// <para>A links data parts.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="linksIndexParts">
57     /// <para>A links index parts.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="header">
61     /// <para>A header.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected
66     ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
67     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
68     ↪ linksIndexParts, LinksHeader<TLink>* header)
69     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
70     {
71         LinksDataParts = linksDataParts;
72         LinksIndexParts = linksIndexParts;
73         Header = header;
74     }
75
76     /// <summary>
77     /// <para>
78     /// Gets the zero.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <returns>
83     /// <para>The link</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override TLink GetZero() => 0U;
88
89     /// <summary>
90     /// <para>
91     /// Determines whether this instance equal to zero.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="value">
96     /// <para>The value.</para>
97     /// <para></para>
98     /// </param>
99     /// <returns>
100    /// <para>The bool</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override bool EqualToZero(TLink value) => value == 0U;
105
106    /// <summary>
107    /// <para>
108    /// Determines whether this instance are equal.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="first">
113    /// <para>The first.</para>
114    /// <para></para>
115    /// </param>

```

```

112     /// <param name="second">
113     /// <para>The second.</para>
114     /// <para></para>
115     /// </param>
116     /// <returns>
117     /// <para>The bool</para>
118     /// <para></para>
119     /// </returns>
120     [MethodImpl(MethodImplOptions.AggressiveInlining)]
121     protected override bool AreEqual(TLink first, TLink second) => first == second;
122
123     /// <summary>
124     /// <para>
125     /// Determines whether this instance greater than zero.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="value">
130     /// <para>The value.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The bool</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected override bool GreaterThanZero(TLink value) => value > 0U;
139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance greater than.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="first">
147     /// <para>The first.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="second">
151     /// <para>The second.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The bool</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override bool GreaterThan(TLink first, TLink second) => first > second;
160
161     /// <summary>
162     /// <para>
163     /// Determines whether this instance greater or equal than.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="first">
168     /// <para>The first.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="second">
172     /// <para>The second.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
181
182     /// <summary>
183     /// <para>
184     /// Determines whether this instance greater or equal than zero.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="value">
189     /// <para>The value.</para>

```

```

190    /// <para></para>
191    /// </param>
192    /// <returns>
193    /// <para>The bool</para>
194    /// <para></para>
195    /// </returns>
196    [MethodImpl(MethodImplOptions.AggressiveInlining)]
197    protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
    ↪ always true for ulong
198
199    /// <summary>
200    /// <para>
201    /// Determines whether this instance less or equal than zero.
202    /// </para>
203    /// <para></para>
204    /// </summary>
205    /// <param name="value">
206    /// <para>The value.</para>
207    /// <para></para>
208    /// </param>
209    /// <returns>
210    /// <para>The bool</para>
211    /// <para></para>
212    /// </returns>
213    [MethodImpl(MethodImplOptions.AggressiveInlining)]
214    protected override bool LessOrEqualThanZero(TLink value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
215
216    /// <summary>
217    /// <para>
218    /// Determines whether this instance less or equal than.
219    /// </para>
220    /// <para></para>
221    /// </summary>
222    /// <param name="first">
223    /// <para>The first.</para>
224    /// <para></para>
225    /// </param>
226    /// <param name="second">
227    /// <para>The second.</para>
228    /// <para></para>
229    /// </param>
230    /// <returns>
231    /// <para>The bool</para>
232    /// <para></para>
233    /// </returns>
234    [MethodImpl(MethodImplOptions.AggressiveInlining)]
235    protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
236
237    /// <summary>
238    /// <para>
239    /// Determines whether this instance less than zero.
240    /// </para>
241    /// <para></para>
242    /// </summary>
243    /// <param name="value">
244    /// <para>The value.</para>
245    /// <para></para>
246    /// </param>
247    /// <returns>
248    /// <para>The bool</para>
249    /// <para></para>
250    /// </returns>
251    [MethodImpl(MethodImplOptions.AggressiveInlining)]
252    protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↪ for ulong
253
254    /// <summary>
255    /// <para>
256    /// Determines whether this instance less than.
257    /// </para>
258    /// <para></para>
259    /// </summary>
260    /// <param name="first">
261    /// <para>The first.</para>
262    /// <para></para>
263    /// </param>
264    /// <param name="second">

```

```

265    /// <para>The second.</para>
266    /// <para></para>
267    /// </param>
268    /// <returns>
269    /// <para>The bool</para>
270    /// <para></para>
271    /// </returns>
272    [MethodImpl(MethodImplOptions.AggressiveInlining)]
273    protected override bool LessThan(TLink first, TLink second) => first < second;
274
275    /// <summary>
276    /// <para>
277    /// Increments the value.
278    /// </para>
279    /// <para></para>
280    /// </summary>
281    /// <param name="value">
282    /// <para>The value.</para>
283    /// <para></para>
284    /// </param>
285    /// <returns>
286    /// <para>The link</para>
287    /// <para></para>
288    /// </returns>
289    [MethodImpl(MethodImplOptions.AggressiveInlining)]
290    protected override TLink Increment(TLink value) => ++value;
291
292    /// <summary>
293    /// <para>
294    /// Decrements the value.
295    /// </para>
296    /// <para></para>
297    /// </summary>
298    /// <param name="value">
299    /// <para>The value.</para>
300    /// <para></para>
301    /// </param>
302    /// <returns>
303    /// <para>The link</para>
304    /// <para></para>
305    /// </returns>
306    [MethodImpl(MethodImplOptions.AggressiveInlining)]
307    protected override TLink Decrement(TLink value) => --value;
308
309    /// <summary>
310    /// <para>
311    /// Adds the first.
312    /// </para>
313    /// <para></para>
314    /// </summary>
315    /// <param name="first">
316    /// <para>The first.</para>
317    /// <para></para>
318    /// </param>
319    /// <param name="second">
320    /// <para>The second.</para>
321    /// <para></para>
322    /// </param>
323    /// <returns>
324    /// <para>The link</para>
325    /// <para></para>
326    /// </returns>
327    [MethodImpl(MethodImplOptions.AggressiveInlining)]
328    protected override TLink Add(TLink first, TLink second) => first + second;
329
330    /// <summary>
331    /// <para>
332    /// Subtracts the first.
333    /// </para>
334    /// <para></para>
335    /// </summary>
336    /// <param name="first">
337    /// <para>The first.</para>
338    /// <para></para>
339    /// </param>
340    /// <param name="second">
341    /// <para>The second.</para>
342    /// <para></para>

```

```

343     /// </param>
344     /// <returns>
345     /// <para>The link</para>
346     /// <para></para>
347     /// </returns>
348     [MethodImpl(MethodImplOptions.AggressiveInlining)]
349     protected override TLink Subtract(TLink first, TLink second) => first - second;
350
351     /// <summary>
352     /// <para>
353     /// Gets the header reference.
354     /// </para>
355     /// <para></para>
356     /// </summary>
357     /// <returns>
358     /// <para>A ref links header of t link</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364     /// <summary>
365     /// <para>
366     /// Gets the link data part reference using the specified link.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380     ↪ ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398     ↪ ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="first">
407     /// <para>The first.</para>
408     /// <para></para>
409     /// </param>
410     /// <param name="second">
411     /// <para>The second.</para>
412     /// <para></para>
413     /// </param>
414     /// <returns>
415     /// <para>The bool</para>
416     /// <para></para>
417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
420     {

```

```

419         ref var firstLink = ref LinksDataParts[first];
420         ref var secondLink = ref LinksDataParts[second];
421         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
422     }
423
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance first is to the right of second.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
448     }
449 }
450 }

```

1.51 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 external links size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16     /// <seealso cref="ILinksTreeMethods{TLink}"/>
17     public unsafe abstract class UInt32ExternalLinksSizeBalancedTreeMethodsBase :
        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The links data parts.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26         /// <summary>
27         /// <para>
28         /// The links index parts.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33         /// <summary>
34         /// <para>
35         /// The header.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected new readonly LinksHeader<TLink>* Header;
40
41         /// <summary>

```

```

42     /// <para>
43     /// Initializes a new <see cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
44     /// instance.
45     /// </para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// </param>
53     /// <param name="linksIndexParts">
54     /// <para>A links index parts.</para>
55     /// </param>
56     /// <param name="header">
57     /// <para>A header.</para>
58     /// </param>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
61     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
62     ↪ linksIndexParts, LinksHeader<TLink>* header)
63     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64     {
65         LinksDataParts = linksDataParts;
66         LinksIndexParts = linksIndexParts;
67         Header = header;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Gets the zero.
73     /// </para>
74     /// </summary>
75     /// <returns>
76     /// <para>The link</para>
77     /// </returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override TLink GetZero() => 0U;
80
81     /// <summary>
82     /// <para>
83     /// Determines whether this instance equal to zero.
84     /// </para>
85     /// </summary>
86     /// <param name="value">
87     /// <para>The value.</para>
88     /// </param>
89     /// <returns>
90     /// <para>The bool</para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected override bool EqualToZero(TLink value) => value == 0U;
94
95     /// <summary>
96     /// <para>
97     /// Determines whether this instance are equal.
98     /// </para>
99     /// </summary>
100    /// <param name="first">
101    /// <para>The first.</para>
102    /// </param>
103    /// <param name="second">
104    /// <para>The second.</para>
105    /// </param>
106    /// <returns>

```

```

117     /// <para>The bool</para>
118     /// <para></para>
119     /// </returns>
120     [MethodImpl(MethodImplOptions.AggressiveInlining)]
121     protected override bool AreEqual(TLink first, TLink second) => first == second;
122
123     /// <summary>
124     /// <para>
125     /// Determines whether this instance greater than zero.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="value">
130     /// <para>The value.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The bool</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected override bool GreaterThanZero(TLink value) => value > OU;
139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance greater than.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="first">
147     /// <para>The first.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="second">
151     /// <para>The second.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The bool</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override bool GreaterThan(TLink first, TLink second) => first > second;
160
161     /// <summary>
162     /// <para>
163     /// Determines whether this instance greater or equal than.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="first">
168     /// <para>The first.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="second">
172     /// <para>The second.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
181
182     /// <summary>
183     /// <para>
184     /// Determines whether this instance greater or equal than zero.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="value">
189     /// <para>The value.</para>
190     /// <para></para>
191     /// </param>
192     /// <returns>
193     /// <para>The bool</para>
194     /// <para></para>

```



```

195     /// </returns>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
    ↪ always true for ulong
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance less or equal than zero.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="value">
206     /// <para>The value.</para>
207     /// <para></para>
208     /// </param>
209     /// <returns>
210     /// <para>The bool</para>
211     /// <para></para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     protected override bool LessOrEqualThanZero(TLink value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
215
216     /// <summary>
217     /// <para>
218     /// Determines whether this instance less or equal than.
219     /// </para>
220     /// <para></para>
221     /// </summary>
222     /// <param name="first">
223     /// <para>The first.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="second">
227     /// <para>The second.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
236
237     /// <summary>
238     /// <para>
239     /// Determines whether this instance less than zero.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="value">
244     /// <para>The value.</para>
245     /// <para></para>
246     /// </param>
247     /// <returns>
248     /// <para>The bool</para>
249     /// <para></para>
250     /// </returns>
251     [MethodImpl(MethodImplOptions.AggressiveInlining)]
252     protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↪ for ulong
253
254     /// <summary>
255     /// <para>
256     /// Determines whether this instance less than.
257     /// </para>
258     /// <para></para>
259     /// </summary>
260     /// <param name="first">
261     /// <para>The first.</para>
262     /// <para></para>
263     /// </param>
264     /// <param name="second">
265     /// <para>The second.</para>
266     /// <para></para>
267     /// </param>
268     /// <returns>
269     /// <para>The bool</para>

```

```

270    /// <para></para>
271    /// </returns>
272    [MethodImpl(MethodImplOptions.AggressiveInlining)]
273    protected override bool LessThan(TLink first, TLink second) => first < second;
274
275    /// <summary>
276    /// <para>
277    /// Increments the value.
278    /// </para>
279    /// <para></para>
280    /// </summary>
281    /// <param name="value">
282    /// <para>The value.</para>
283    /// <para></para>
284    /// </param>
285    /// <returns>
286    /// <para>The link</para>
287    /// <para></para>
288    /// </returns>
289    [MethodImpl(MethodImplOptions.AggressiveInlining)]
290    protected override TLink Increment(TLink value) => ++value;
291
292    /// <summary>
293    /// <para>
294    /// Decrements the value.
295    /// </para>
296    /// <para></para>
297    /// </summary>
298    /// <param name="value">
299    /// <para>The value.</para>
300    /// <para></para>
301    /// </param>
302    /// <returns>
303    /// <para>The link</para>
304    /// <para></para>
305    /// </returns>
306    [MethodImpl(MethodImplOptions.AggressiveInlining)]
307    protected override TLink Decrement(TLink value) => --value;
308
309    /// <summary>
310    /// <para>
311    /// Adds the first.
312    /// </para>
313    /// <para></para>
314    /// </summary>
315    /// <param name="first">
316    /// <para>The first.</para>
317    /// <para></para>
318    /// </param>
319    /// <param name="second">
320    /// <para>The second.</para>
321    /// <para></para>
322    /// </param>
323    /// <returns>
324    /// <para>The link</para>
325    /// <para></para>
326    /// </returns>
327    [MethodImpl(MethodImplOptions.AggressiveInlining)]
328    protected override TLink Add(TLink first, TLink second) => first + second;
329
330    /// <summary>
331    /// <para>
332    /// Subtracts the first.
333    /// </para>
334    /// <para></para>
335    /// </summary>
336    /// <param name="first">
337    /// <para>The first.</para>
338    /// <para></para>
339    /// </param>
340    /// <param name="second">
341    /// <para>The second.</para>
342    /// <para></para>
343    /// </param>
344    /// <returns>
345    /// <para>The link</para>
346    /// <para></para>
347    /// </returns>

```

```

348 [MethodImpl(MethodImplOptions.AggressiveInlining)]
349 protected override TLink Subtract(TLink first, TLink second) => first - second;
350
351 /// <summary>
352 /// <para>
353 /// Gets the header reference.
354 /// </para>
355 /// <para></para>
356 /// </summary>
357 /// <returns>
358 /// <para>A ref links header of t link</para>
359 /// <para></para>
360 /// </returns>
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364 /// <summary>
365 /// <para>
366 /// Gets the link data part reference using the specified link.
367 /// </para>
368 /// <para></para>
369 /// </summary>
370 /// <param name="link">
371 /// <para>The link.</para>
372 /// <para></para>
373 /// </param>
374 /// <returns>
375 /// <para>A ref raw link data part of t link</para>
376 /// <para></para>
377 /// </returns>
378 [MethodImpl(MethodImplOptions.AggressiveInlining)]
379 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
    ↪ ref LinksDataParts[link];
380
381 /// <summary>
382 /// <para>
383 /// Gets the link index part reference using the specified link.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 /// <param name="link">
388 /// <para>The link.</para>
389 /// <para></para>
390 /// </param>
391 /// <returns>
392 /// <para>A ref raw link index part of t link</para>
393 /// <para></para>
394 /// </returns>
395 [MethodImpl(MethodImplOptions.AggressiveInlining)]
396 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↪ ref LinksIndexParts[link];
397
398 /// <summary>
399 /// <para>
400 /// Determines whether this instance first is to the left of second.
401 /// </para>
402 /// <para></para>
403 /// </summary>
404 /// <param name="first">
405 /// <para>The first.</para>
406 /// <para></para>
407 /// </param>
408 /// <param name="second">
409 /// <para>The second.</para>
410 /// <para></para>
411 /// </param>
412 /// <returns>
413 /// <para>The bool</para>
414 /// <para></para>
415 /// </returns>
416 [MethodImpl(MethodImplOptions.AggressiveInlining)]
417 protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
418 {
419     ref var firstLink = ref LinksDataParts[first];
420     ref var secondLink = ref LinksDataParts[second];
421     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
422 }

```

```

423     /// <summary>
424     /// <para>
425     /// Determines whether this instance first is to the right of second.
426     /// </para>
427     /// <para></para>
428     /// </summary>
429     /// <param name="first">
430     /// <para>The first.</para>
431     /// <para></para>
432     /// </param>
433     /// <param name="second">
434     /// <para>The second.</para>
435     /// <para></para>
436     /// </param>
437     /// <returns>
438     /// <para>The bool</para>
439     /// <para></para>
440     /// </returns>
441     [MethodImpl(MethodImplOptions.AggressiveInlining)]
442     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
443     {
444         ref var firstLink = ref LinksDataParts[first];
445         ref var secondLink = ref LinksDataParts[second];
446         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
447             ↪ secondLink.Source, secondLink.Target);
448     }
449 }
450 }

```

1.52 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16     ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public
43         ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46         ↪ linksIndexParts, header) { }

```

```

42    /// <summary>
43    /// <para>
44    /// Gets the left reference using the specified node.
45    /// </para>
46    /// <para></para>
47    /// </summary>
48    /// <param name="node">
49    /// <para>The node.</para>
50    /// <para></para>
51    /// </param>
52    /// <returns>
53    /// <para>The ref link</para>
54    /// <para></para>
55    /// </returns>
56    [MethodImpl(MethodImplOptions.AggressiveInlining)]
57    protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;
58
59    /// <summary>
60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// <para></para>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// <para></para>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// <para></para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>

```

```

118     /// <para></para>
119     /// </param>
120     /// <param name="left">
121     /// <para>The left.</para>
122     /// <para></para>
123     /// </param>
124     [MethodImpl(MethodImplOptions.AggressiveInlining)]
125     protected override void SetLeft(TLink node, TLink left) =>
126         ↪ LinksIndexParts[node].LeftAsSource = left;
127
128     /// <summary>
129     /// <para>
130     /// Sets the right using the specified node.
131     /// </para>
132     /// <para></para>
133     /// </summary>
134     /// <param name="node">
135     /// <para>The node.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="right">
139     /// <para>The right.</para>
140     /// <para></para>
141     /// </param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected override void SetRight(TLink node, TLink right) =>
144         ↪ LinksIndexParts[node].RightAsSource = right;
145
146     /// <summary>
147     /// <para>
148     /// Gets the size using the specified node.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="node">
153     /// <para>The node.</para>
154     /// <para></para>
155     /// </param>
156     /// <returns>
157     /// <para>The link</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
162
163     /// <summary>
164     /// <para>
165     /// Sets the size using the specified node.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="node">
170     /// <para>The node.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="size">
174     /// <para>The size.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetSize(TLink node, TLink size) =>
179         ↪ LinksIndexParts[node].SizeAsSource = size;
180
181     /// <summary>
182     /// <para>
183     /// Gets the tree root.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <returns>
188     /// <para>The link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected override TLink GetTreeRoot() => Header->RootAsSource;

```

```

193     /// Gets the base part value using the specified node.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="node">
198     /// <para>The node.</para>
199     /// <para></para>
200     /// </param>
201     /// <returns>
202     /// <para>The link</para>
203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance first is to the left of second.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236     ↪ TLink secondSource, TLink secondTarget)
237     => firstSource < secondSource || firstSource == secondSource && firstTarget <
238     ↪ secondTarget;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268     ↪ TLink secondSource, TLink secondTarget)
269     => firstSource > secondSource || firstSource == secondSource && firstTarget >
270     ↪ secondTarget;

```

```

267
268     /// <summary>
269     /// <para>
270     /// Clears the node using the specified node.
271     /// </para>
272     /// <para></para>
273     /// </summary>
274     /// <param name="node">
275     /// <para>The node.</para>
276     /// <para></para>
277     /// </param>
278     [MethodImpl(MethodImplOptions.AggressiveInlining)]
279     protected override void ClearNode(TLink node)
280     {
281         ref var link = ref LinksIndexParts[node];
282         link.LeftAsSource = Zero;
283         link.RightAsSource = Zero;
284         link.SizeAsSource = Zero;
285     }
286 }
287 }

```

1.53 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksSourcesSizeBalancedTreeMethods :
16     ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32ExternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt32ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
43         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45         ↪ linksIndexParts, header) { }
46
47         /// <summary>
48         /// <para>
49         /// Gets the left reference using the specified node.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="node">
54         /// <para>The node.</para>
55         /// <para></para>
56         /// </param>

```



```

52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
58         ↳ LinksIndexParts[node].LeftAsSource;
59
60     /// <summary>
61     /// <para>
62     /// Gets the right reference using the specified node.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="node">
67     /// <para>The node.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The ref link</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected override ref TLink GetRightReference(TLink node) => ref
76         ↳ LinksIndexParts[node].RightAsSource;
77
78     /// <summary>
79     /// <para>
80     /// Gets the left using the specified node.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="node">
85     /// <para>The node.</para>
86     /// <para></para>
87     /// </param>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
94
95     /// <summary>
96     /// <para>
97     /// Gets the right using the specified node.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="node">
102    /// <para>The node.</para>
103    /// <para></para>
104    /// </param>
105    /// <returns>
106    /// <para>The link</para>
107    /// <para></para>
108    /// </returns>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLink node, TLink left) =>
128        ↳ LinksIndexParts[node].LeftAsSource = left;

```

```

127     /// <summary>
128     /// <para>
129     /// Sets the right using the specified node.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].RightAsSource = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="node">
152     /// <para>The node.</para>
153     /// <para></para>
154     /// </param>
155     /// <returns>
156     /// <para>The link</para>
157     /// <para></para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLink node, TLink size) =>
178         ↳ LinksIndexParts[node].SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLink GetTreeRoot() => Header->RootAsSource;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>

```

```

203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance first is to the left of second.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236     ↪ TLink secondSource, TLink secondTarget)
237     => firstSource < secondSource || firstSource == secondSource && firstTarget <
238     ↪ secondTarget;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268     ↪ TLink secondSource, TLink secondTarget)
269     => firstSource > secondSource || firstSource == secondSource && firstTarget >
270     ↪ secondTarget;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>

```

```

277     /// </param>
278     [MethodImpl(MethodImplOptions.AggressiveInlining)]
279     protected override void ClearNode(TLink node)
280     {
281         ref var link = ref LinksIndexParts[node];
282         link.LeftAsSource = Zero;
283         link.RightAsSource = Zero;
284         link.SizeAsSource = Zero;
285     }
286 }
287 }

```

1.54 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16     ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public
43         ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46         ↪ linksIndexParts, header) { }
47
48         /// <summary>
49         /// <para>
50         /// Gets the left reference using the specified node.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="node">
55         /// <para>The node.</para>
56         /// <para></para>
57         /// </param>
58         /// <returns>
59         /// <para>The ref link</para>
60         /// <para></para>
61         /// </returns>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         protected override ref TLink GetLeftReference(TLink node) => ref
64         ↪ LinksIndexParts[node].LeftAsTarget;
65
66         /// <summary>

```

```

60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// <para></para>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// <para></para>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// <para></para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ LinksIndexParts[node].RightAsTarget;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
    ↪ LinksIndexParts[node].LeftAsTarget = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">
134    /// <para>The node.</para>
135    /// <para></para>

```

```

136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].RightAsTarget = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="node">
152     /// <para>The node.</para>
153     /// <para></para>
154     /// </param>
155     /// <returns>
156     /// <para>The link</para>
157     /// <para></para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLink node, TLink size) =>
178         ↳ LinksIndexParts[node].SizeAsTarget = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLink GetTreeRoot() => Header->RootAsTarget;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance first is to the left of second.
213     /// </para>

```

```

212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236     ↪ TLink secondSource, TLink secondTarget)
237     => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
238     ↪ secondSource;
239
240     /// <summary>
241     /// <para>
242     /// <para>Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268     ↪ TLink secondSource, TLink secondTarget)
269     => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
270     ↪ secondSource;
271
272     /// <summary>
273     /// <para>
274     /// <para>Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsTarget = Zero;
287         link.RightAsTarget = Zero;
288         link.SizeAsTarget = Zero;
289     }

```

```
286     }
287 }
```

1.55 ./csharp/Platform.Data.Doublets.Memory.Split.Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods

```
1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt32;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 external links targets size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt32ExternalLinksTargetsSizeBalancedTreeMethods :
16        ↳ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
21        ↳ instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public UInt32ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
43            ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44            ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45            ↳ linksIndexParts, header) { }
46
47        /// <summary>
48        /// <para>
49        /// Gets the left reference using the specified node.
50        /// </para>
51        /// <para></para>
52        /// </summary>
53        /// <param name="node">
54        /// <para>The node.</para>
55        /// <para></para>
56        /// </param>
57        /// <returns>
58        /// <para>The ref link</para>
59        /// <para></para>
60        /// </returns>
61        [MethodImpl(MethodImplOptions.AggressiveInlining)]
62        protected override ref TLink GetLeftReference(TLink node) => ref
63            ↳ LinksIndexParts[node].LeftAsTarget;
64
65        /// <summary>
66        /// <para>
67        /// Gets the right reference using the specified node.
68        /// </para>
69        /// <para></para>
70        /// </summary>
71        /// <param name="node">
72        /// <para>The node.</para>
73        /// <para></para>
74        /// </param>
```



```

69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
75         ↳ LinksIndexParts[node].RightAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// <para></para>
120    /// </param>
121    /// <param name="left">
122    /// <para>The left.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected override void SetLeft(TLink node, TLink left) =>
127        ↳ LinksIndexParts[node].LeftAsTarget = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>
141    /// <para></para>
142    /// </param>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void SetRight(TLink node, TLink right) =>
145        ↳ LinksIndexParts[node].RightAsTarget = right;

```

```

144    /// <summary>
145    /// <para>
146    /// Gets the size using the specified node.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="node">
151    /// <para>The node.</para>
152    /// <para></para>
153    /// </param>
154    /// <returns>
155    /// <para>The link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
160
161    /// <summary>
162    /// <para>
163    /// Sets the size using the specified node.
164    /// </para>
165    /// <para></para>
166    /// </summary>
167    /// <param name="node">
168    /// <para>The node.</para>
169    /// <para></para>
170    /// </param>
171    /// <param name="size">
172    /// <para>The size.</para>
173    /// <para></para>
174    /// </param>
175    [MethodImpl(MethodImplOptions.AggressiveInlining)]
176    protected override void SetSize(TLink node, TLink size) =>
177        ↳ LinksIndexParts[node].SizeAsTarget = size;
178
179    /// <summary>
180    /// <para>
181    /// Gets the tree root.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <returns>
186    /// <para>The link</para>
187    /// <para></para>
188    /// </returns>
189    [MethodImpl(MethodImplOptions.AggressiveInlining)]
190    protected override TLink GetTreeRoot() => Header->RootAsTarget;
191
192    /// <summary>
193    /// <para>
194    /// Gets the base part value using the specified node.
195    /// </para>
196    /// <para></para>
197    /// </summary>
198    /// <param name="node">
199    /// <para>The node.</para>
200    /// <para></para>
201    /// </param>
202    /// <returns>
203    /// <para>The link</para>
204    /// <para></para>
205    /// </returns>
206    [MethodImpl(MethodImplOptions.AggressiveInlining)]
207    protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance first is to the left of second.
212    /// </para>
213    /// <para></para>
214    /// </summary>
215    /// <param name="firstSource">
216    /// <para>The first source.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="firstTarget">
220    /// <para>The first target.</para>
221    /// <para></para>
222    /// </param>

```

```

221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↪ TLink secondSource, TLink secondTarget)
237         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
238         ↪ secondSource;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268         ↪ TLink secondSource, TLink secondTarget)
269         => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
270         ↪ secondSource;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsTarget = Zero;
287         link.RightAsTarget = Zero;
288         link.SizeAsTarget = Zero;
289     }
290 }

```

1.56 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeM

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

```

```

6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 internal links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16    public unsafe abstract class UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26        /// <summary>
27        /// <para>
28        /// The links index parts.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33        /// <summary>
34        /// <para>
35        /// The header.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        protected new readonly LinksHeader<TLink>* Header;
40
41        /// <summary>
42        /// <para>
43        /// Initializes a new <see
44        ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45        /// </para>
46        /// <para></para>
47        /// </summary>
48        /// <param name="constants">
49        /// <para>A constants.</para>
50        /// <para></para>
51        /// </param>
52        /// <param name="linksDataParts">
53        /// <para>A links data parts.</para>
54        /// <para></para>
55        /// </param>
56        /// <param name="linksIndexParts">
57        /// <para>A links index parts.</para>
58        /// <para></para>
59        /// </param>
60        /// <param name="header">
61        /// <para>A header.</para>
62        /// <para></para>
63        /// </param>
64        [MethodImpl(MethodImplOptions.AggressiveInlining)]
65        protected
66        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
67        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
68        ↪ linksIndexParts, LinksHeader<TLink>* header)
69        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
70        {
71            LinksDataParts = linksDataParts;
72            LinksIndexParts = linksIndexParts;
73            Header = header;
74        }
75
76        /// <summary>
77        /// <para>
78        /// Gets the zero.
79        /// </para>
80        /// <para></para>
81        /// </summary>
82        /// <returns>
83        /// <para>The link</para>

```

```

79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override TLink GetZero() => OU;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(TLink value) => value == OU;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(TLink first, TLink second) => first == second;
121
122     /// <summary>
123     /// <para>
124     /// Determines whether this instance greater than zero.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The bool</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override bool GreaterThanZero(TLink value) => value > OU;
138
139     /// <summary>
140     /// <para>
141     /// Determines whether this instance greater than.
142     /// </para>
143     /// <para></para>
144     /// </summary>
145     /// <param name="first">
146     /// <para>The first.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="second">
150     /// <para>The second.</para>
151     /// <para></para>
152     /// </param>
153     /// <returns>
154     /// <para>The bool</para>
155     /// <para></para>
156     /// </returns>

```

```

157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 protected override bool GreaterThan(TLink first, TLink second) => first > second;
159
160 /// <summary>
161 /// <para>
162 /// Determines whether this instance greater or equal than.
163 /// </para>
164 /// <para></para>
165 /// </summary>
166 /// <param name="first">
167 /// <para>The first.</para>
168 /// <para></para>
169 /// </param>
170 /// <param name="second">
171 /// <para>The second.</para>
172 /// <para></para>
173 /// </param>
174 /// <returns>
175 /// <para>The bool</para>
176 /// <para></para>
177 /// </returns>
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
180
181 /// <summary>
182 /// <para>
183 /// Determines whether this instance greater or equal than zero.
184 /// </para>
185 /// <para></para>
186 /// </summary>
187 /// <param name="value">
188 /// <para>The value.</para>
189 /// <para></para>
190 /// </param>
191 /// <returns>
192 /// <para>The bool</para>
193 /// <para></para>
194 /// </returns>
195 [MethodImpl(MethodImplOptions.AggressiveInlining)]
196 protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
    ↳ always true for ulong
197
198 /// <summary>
199 /// <para>
200 /// Determines whether this instance less or equal than zero.
201 /// </para>
202 /// <para></para>
203 /// </summary>
204 /// <param name="value">
205 /// <para>The value.</para>
206 /// <para></para>
207 /// </param>
208 /// <returns>
209 /// <para>The bool</para>
210 /// <para></para>
211 /// </returns>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 protected override bool LessOrEqualThanZero(TLink value) => value == OUL; // value is
    ↳ always >= 0 for ulong
214
215 /// <summary>
216 /// <para>
217 /// Determines whether this instance less or equal than.
218 /// </para>
219 /// <para></para>
220 /// </summary>
221 /// <param name="first">
222 /// <para>The first.</para>
223 /// <para></para>
224 /// </param>
225 /// <param name="second">
226 /// <para>The second.</para>
227 /// <para></para>
228 /// </param>
229 /// <returns>
230 /// <para>The bool</para>
231 /// <para></para>
232 /// </returns>

```

```

233 [MethodImpl(MethodImplOptions.AggressiveInlining)]
234 protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
235
236 /// <summary>
237 /// <para>
238 /// Determines whether this instance less than zero.
239 /// </para>
240 /// <para></para>
241 /// </summary>
242 /// <param name="value">
243 /// <para>The value.</para>
244 /// <para></para>
245 /// </param>
246 /// <returns>
247 /// <para>The bool</para>
248 /// <para></para>
249 /// </returns>
250 [MethodImpl(MethodImplOptions.AggressiveInlining)]
251 protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↳ for ulong
252
253 /// <summary>
254 /// <para>
255 /// Determines whether this instance less than.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="first">
260 /// <para>The first.</para>
261 /// <para></para>
262 /// </param>
263 /// <param name="second">
264 /// <para>The second.</para>
265 /// <para></para>
266 /// </param>
267 /// <returns>
268 /// <para>The bool</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 protected override bool LessThan(TLink first, TLink second) => first < second;
273
274 /// <summary>
275 /// <para>
276 /// Increments the value.
277 /// </para>
278 /// <para></para>
279 /// </summary>
280 /// <param name="value">
281 /// <para>The value.</para>
282 /// <para></para>
283 /// </param>
284 /// <returns>
285 /// <para>The link</para>
286 /// <para></para>
287 /// </returns>
288 [MethodImpl(MethodImplOptions.AggressiveInlining)]
289 protected override TLink Increment(TLink value) => ++value;
290
291 /// <summary>
292 /// <para>
293 /// Decrements the value.
294 /// </para>
295 /// <para></para>
296 /// </summary>
297 /// <param name="value">
298 /// <para>The value.</para>
299 /// <para></para>
300 /// </param>
301 /// <returns>
302 /// <para>The link</para>
303 /// <para></para>
304 /// </returns>
305 [MethodImpl(MethodImplOptions.AggressiveInlining)]
306 protected override TLink Decrement(TLink value) => --value;
307
308 /// <summary>
309 /// <para>

```

```

310     /// Adds the first.
311     /// </para>
312     /// <para></para>
313     /// </summary>
314     /// <param name="first">
315     /// <para>The first.</para>
316     /// <para></para>
317     /// </param>
318     /// <param name="second">
319     /// <para>The second.</para>
320     /// <para></para>
321     /// </param>
322     /// <returns>
323     /// <para>The link</para>
324     /// <para></para>
325     /// </returns>
326     [MethodImpl(MethodImplOptions.AggressiveInlining)]
327     protected override TLink Add(TLink first, TLink second) => first + second;
328
329     /// <summary>
330     /// <para>
331     /// Subtracts the first.
332     /// </para>
333     /// <para></para>
334     /// </summary>
335     /// <param name="first">
336     /// <para>The first.</para>
337     /// <para></para>
338     /// </param>
339     /// <param name="second">
340     /// <para>The second.</para>
341     /// <para></para>
342     /// </param>
343     /// <returns>
344     /// <para>The link</para>
345     /// <para></para>
346     /// </returns>
347     [MethodImpl(MethodImplOptions.AggressiveInlining)]
348     protected override TLink Subtract(TLink first, TLink second) => first - second;
349
350     /// <summary>
351     /// <para>
352     /// Gets the link data part reference using the specified link.
353     /// </para>
354     /// <para></para>
355     /// </summary>
356     /// <param name="link">
357     /// <para>The link.</para>
358     /// <para></para>
359     /// </param>
360     /// <returns>
361     /// <para>A ref raw link data part of t link</para>
362     /// <para></para>
363     /// </returns>
364     [MethodImpl(MethodImplOptions.AggressiveInlining)]
365     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
366     ↪ ref LinksDataParts[link];
367
368     /// <summary>
369     /// <para>
370     /// Gets the link index part reference using the specified link.
371     /// </para>
372     /// <para></para>
373     /// </summary>
374     /// <param name="link">
375     /// <para>The link.</para>
376     /// <para></para>
377     /// </param>
378     /// <returns>
379     /// <para>A ref raw link index part of t link</para>
380     /// <para></para>
381     /// </returns>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
384     ↪ ref LinksIndexParts[link];
385
386     /// <summary>
387     /// <para>

```



```

386     /// Determines whether this instance first is to the left of second.
387     /// </para>
388     /// <para></para>
389     /// </summary>
390     /// <param name="first">
391     /// <para>The first.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="second">
395     /// <para>The second.</para>
396     /// <para></para>
397     /// </param>
398     /// <returns>
399     /// <para>The bool</para>
400     /// <para></para>
401     /// </returns>
402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
403     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
404         ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
405     /// <summary>
406     /// <para>
407     /// Determines whether this instance first is to the right of second.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="first">
412     /// <para>The first.</para>
413     /// <para></para>
414     /// </param>
415     /// <param name="second">
416     /// <para>The second.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
425         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
426 }

```

1.57 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 internal links size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16     public unsafe abstract class UInt32InternalLinksSizeBalancedTreeMethodsBase :
17         ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The links data parts.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26         /// <summary>
27         /// <para>
28         /// The links index parts.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33     }

```

```

33     /// <para>
34     /// The header.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected new readonly LinksHeader<TLink>* Header;
39
40     /// <summary>
41     /// <para>
42     /// Initializes a new <see cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
43     ↪ instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="linksDataParts">
52     /// <para>A links data parts.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="linksIndexParts">
56     /// <para>A links index parts.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
65     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
66     ↪ linksIndexParts, LinksHeader<TLink>* header)
67     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68     {
69         LinksDataParts = linksDataParts;
70         LinksIndexParts = linksIndexParts;
71         Header = header;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Gets the zero.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <returns>
81     /// <para>The link</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override TLink GetZero() => 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance equal to zero.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="value">
94     /// <para>The value.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The bool</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override bool EqualToZero(TLink value) => value == 0U;
103
104    /// <summary>
105    /// <para>
106    /// Determines whether this instance are equal.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="first">

```

```

108    /// <para>The first.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="second">
112    /// <para>The second.</para>
113    /// <para></para>
114    /// </param>
115    /// <returns>
116    /// <para>The bool</para>
117    /// <para></para>
118    /// </returns>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override bool AreEqual(TLink first, TLink second) => first == second;
121
122    /// <summary>
123    /// <para>
124    /// Determines whether this instance greater than zero.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="value">
129    /// <para>The value.</para>
130    /// <para></para>
131    /// </param>
132    /// <returns>
133    /// <para>The bool</para>
134    /// <para></para>
135    /// </returns>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    protected override bool GreaterThanZero(TLink value) => value > 0U;
138
139    /// <summary>
140    /// <para>
141    /// Determines whether this instance greater than.
142    /// </para>
143    /// <para></para>
144    /// </summary>
145    /// <param name="first">
146    /// <para>The first.</para>
147    /// <para></para>
148    /// </param>
149    /// <param name="second">
150    /// <para>The second.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The bool</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override bool GreaterThan(TLink first, TLink second) => first > second;
159
160    /// <summary>
161    /// <para>
162    /// Determines whether this instance greater or equal than.
163    /// </para>
164    /// <para></para>
165    /// </summary>
166    /// <param name="first">
167    /// <para>The first.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="second">
171    /// <para>The second.</para>
172    /// <para></para>
173    /// </param>
174    /// <returns>
175    /// <para>The bool</para>
176    /// <para></para>
177    /// </returns>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
180
181    /// <summary>
182    /// <para>
183    /// Determines whether this instance greater or equal than zero.
184    /// </para>
185    /// <para></para>

```

```

186     /// </summary>
187     /// <param name="value">
188     /// <para>The value.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The bool</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
    ↪ always true for ulong
197
198     /// <summary>
199     /// <para>
200     /// Determines whether this instance less or equal than zero.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <param name="value">
205     /// <para>The value.</para>
206     /// <para></para>
207     /// </param>
208     /// <returns>
209     /// <para>The bool</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     protected override bool LessOrEqualThanZero(TLink value) => value == OUL; // value is
    ↪ always >= 0 for ulong
214
215     /// <summary>
216     /// <para>
217     /// Determines whether this instance less or equal than.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="first">
222     /// <para>The first.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="second">
226     /// <para>The second.</para>
227     /// <para></para>
228     /// </param>
229     /// <returns>
230     /// <para>The bool</para>
231     /// <para></para>
232     /// </returns>
233     [MethodImpl(MethodImplOptions.AggressiveInlining)]
234     protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance less than zero.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="value">
243     /// <para>The value.</para>
244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>The bool</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↪ for ulong
252
253     /// <summary>
254     /// <para>
255     /// Determines whether this instance less than.
256     /// </para>
257     /// <para></para>
258     /// </summary>
259     /// <param name="first">
260     /// <para>The first.</para>

```

```

261    /// <para></para>
262    /// </param>
263    /// <param name="second">
264    /// <para>The second.</para>
265    /// <para></para>
266    /// </param>
267    /// <returns>
268    /// <para>The bool</para>
269    /// <para></para>
270    /// </returns>
271    [MethodImpl(MethodImplOptions.AggressiveInlining)]
272    protected override bool LessThan(TLink first, TLink second) => first < second;
273
274    /// <summary>
275    /// <para>
276    /// Increments the value.
277    /// </para>
278    /// <para></para>
279    /// </summary>
280    /// <param name="value">
281    /// <para>The value.</para>
282    /// <para></para>
283    /// </param>
284    /// <returns>
285    /// <para>The link</para>
286    /// <para></para>
287    /// </returns>
288    [MethodImpl(MethodImplOptions.AggressiveInlining)]
289    protected override TLink Increment(TLink value) => ++value;
290
291    /// <summary>
292    /// <para>
293    /// Decrements the value.
294    /// </para>
295    /// <para></para>
296    /// </summary>
297    /// <param name="value">
298    /// <para>The value.</para>
299    /// <para></para>
300    /// </param>
301    /// <returns>
302    /// <para>The link</para>
303    /// <para></para>
304    /// </returns>
305    [MethodImpl(MethodImplOptions.AggressiveInlining)]
306    protected override TLink Decrement(TLink value) => --value;
307
308    /// <summary>
309    /// <para>
310    /// Adds the first.
311    /// </para>
312    /// <para></para>
313    /// </summary>
314    /// <param name="first">
315    /// <para>The first.</para>
316    /// <para></para>
317    /// </param>
318    /// <param name="second">
319    /// <para>The second.</para>
320    /// <para></para>
321    /// </param>
322    /// <returns>
323    /// <para>The link</para>
324    /// <para></para>
325    /// </returns>
326    [MethodImpl(MethodImplOptions.AggressiveInlining)]
327    protected override TLink Add(TLink first, TLink second) => first + second;
328
329    /// <summary>
330    /// <para>
331    /// Subtracts the first.
332    /// </para>
333    /// <para></para>
334    /// </summary>
335    /// <param name="first">
336    /// <para>The first.</para>
337    /// <para></para>
338    /// </param>

```

```

/// <param name="second">
/// <para>The second.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override TLink Subtract(TLink first, TLink second) => first - second;

/// <summary>
/// <para>
/// Gets the link data part reference using the specified link.
/// </para>
/// <para></para>
/// </summary>
/// <param name="link">
/// <para>The link.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>A ref raw link data part of t link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
    ↪ ref LinksDataParts[link];

/// <summary>
/// <para>
/// Gets the link index part reference using the specified link.
/// </para>
/// <para></para>
/// </summary>
/// <param name="link">
/// <para>The link.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>A ref raw link index part of t link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↪ ref LinksIndexParts[link];

/// <summary>
/// <para>
/// Determines whether this instance first is to the left of second.
/// </para>
/// <para></para>
/// </summary>
/// <param name="first">
/// <para>The first.</para>
/// <para></para>
/// </param>
/// <param name="second">
/// <para>The second.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The bool</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
    ↪ GetKeyPartValue(first) < GetKeyPartValue(second);

/// <summary>
/// <para>
/// Determines whether this instance first is to the right of second.
/// </para>
/// <para></para>
/// </summary>
/// <param name="first">
/// <para>The first.</para>
/// <para></para>
/// </param>

```

```

414     /// </param>
415     /// <param name="second">
416     /// <para>The second.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
        ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

1.58 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Generic
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources linked list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}">
15     public unsafe class UInt32InternalLinksSourcesLinkedListMethods :
        ↪ InternalLinksSourcesLinkedListMethods<TLink>
16     {
17         private readonly RawLinkDataPart<TLink>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="constants">
27         /// <para>A constants.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="linksDataParts">
31         /// <para>A links data parts.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="linksIndexParts">
35         /// <para>A links index parts.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
        ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
40         {
41             _linksDataParts = linksDataParts;
42             _linksIndexParts = linksIndexParts;
43         }
44
45         /// <summary>
46         /// <para>
47         /// Gets the link data part reference using the specified link.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="link">
52         /// <para>The link.</para>
53         /// <para></para>
54         /// </param>
55         /// <returns>
56         /// <para>A ref raw link data part of t link</para>
57         /// <para></para>
58         /// </returns>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60

```

```

61     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪ ref _linksDataParts[link];
62
63     /// <summary>
64     /// <para>
65     /// Gets the link index part reference using the specified link.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="link">
70     /// <para>The link.</para>
71     /// <para></para>
72     /// </param>
73     /// <returns>
74     /// <para>A ref raw link index part of t link</para>
75     /// <para></para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref _linksIndexParts[link];
79 }
80 }

```

1.59 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
        ↪ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public
        ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
        ↪ linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="node">

```



```

49    /// <para>The node.</para>
50    /// <para></para>
51    /// </param>
52    /// <returns>
53    /// <para>The ref link</para>
54    /// <para></para>
55    /// </returns>
56    [MethodImpl(MethodImplOptions.AggressiveInlining)]
57    protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;
58
59    /// <summary>
60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// <para></para>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// <para></para>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// <para></para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

125     protected override void SetLeft(TLink node, TLink left) =>
126         ↳ LinksIndexParts[node].LeftAsSource = left;
127
128     /// <summary>
129     /// <para>
130     /// Sets the right using the specified node.
131     /// </para>
132     /// <para></para>
133     /// </summary>
134     /// <param name="node">
135     /// <para>The node.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="right">
139     /// <para>The right.</para>
140     /// <para></para>
141     /// </param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected override void SetRight(TLink node, TLink right) =>
144         ↳ LinksIndexParts[node].RightAsSource = right;
145
146     /// <summary>
147     /// <para>
148     /// Gets the size using the specified node.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="node">
153     /// <para>The node.</para>
154     /// <para></para>
155     /// </param>
156     /// <returns>
157     /// <para>The link</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
162
163     /// <summary>
164     /// <para>
165     /// Sets the size using the specified node.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="node">
170     /// <para>The node.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="size">
174     /// <para>The size.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetSize(TLink node, TLink size) =>
179         ↳ LinksIndexParts[node].SizeAsSource = size;
180
181     /// <summary>
182     /// <para>
183     /// Gets the tree root using the specified node.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="node">
188     /// <para>The node.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The link</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsSource;
197
198     /// <summary>
199     /// <para>
200     /// Gets the base part value using the specified node.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <param name="node">
205     /// <para>The node.</para>
206     /// <para></para>
207     /// </param>
208     /// <returns>
209     /// <para>The base part value</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     protected override TLink GetBasePart(TLink node) => LinksIndexParts[node].BasePartAsSource;

```

```

200     /// </summary>
201     /// <param name="node">
202     /// <para>The node.</para>
203     /// <para></para>
204     /// </param>
205     /// <returns>
206     /// <para>The link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
211
212     /// <summary>
213     /// <para>
214     /// Gets the key part value using the specified node.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="node">
219     /// <para>The node.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;
228
229     /// <summary>
230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref LinksIndexParts[node];
243         link.LeftAsSource = Zero;
244         link.RightAsSource = Zero;
245         link.SizeAsSource = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(source), target);
268 }

```

1.60 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethod

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt32;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {

```

```

8  /// <summary>
9  /// <para>
10 /// Represents the int 32 internal links sources size balanced tree methods.
11 /// </para>
12 /// <para></para>
13 /// </summary>
14 /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15 public unsafe class UInt32InternalLinksSourcesSizeBalancedTreeMethods :
    ↳ UInt32InternalLinksSizeBalancedTreeMethodsBase
16 {
17     /// <summary>
18     /// <para>
19     /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
    ↳ instance.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     /// <param name="constants">
24     /// <para>A constants.</para>
25     /// <para></para>
26     /// </param>
27     /// <param name="linksDataParts">
28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↳ linksIndexParts, header) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the left reference using the specified node.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ LinksIndexParts[node].LeftAsSource;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
    ↳ LinksIndexParts[node].RightAsSource;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.

```

```

79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↳ LinksIndexParts[node].LeftAsSource = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↳ LinksIndexParts[node].RightAsSource = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>

```

```

155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
160
161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLink node, TLink size) =>
177         ↪ LinksIndexParts[node].SizeAsSource = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root using the specified node.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="node">
186     /// <para>The node.</para>
187     /// <para></para>
188     /// </param>
189     /// <returns>
190     /// <para>The link</para>
191     /// <para></para>
192     /// </returns>
193     [MethodImpl(MethodImplOptions.AggressiveInlining)]
194     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsSource;
195
196     /// <summary>
197     /// <para>
198     /// Gets the base part value using the specified node.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     /// <param name="node">
203     /// <para>The node.</para>
204     /// <para></para>
205     /// </param>
206     /// <returns>
207     /// <para>The link</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
212
213     /// <summary>
214     /// <para>
215     /// Gets the key part value using the specified node.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="node">
220     /// <para>The node.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The link</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;
229
230     /// <summary>
231     /// <para>
232     /// Clears the node using the specified node.

```

```

232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref LinksIndexParts[node];
243         link.LeftAsSource = Zero;
244         link.RightAsSource = Zero;
245         link.SizeAsSource = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(source), target);
268 }

```

1.61 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalance

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt32;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 32 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16         ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>

```

```

37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public
41     ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
43     ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
44     ↪ linksIndexParts, header) { }
45
46     /// <summary>
47     /// <para>
48     /// Gets the left reference using the specified node.
49     /// </para>
50     /// </summary>
51     /// <param name="node">
52     /// <para>The node.</para>
53     /// </param>
54     /// <returns>
55     /// <para>The ref link</para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override ref TLink GetLeftReference(TLink node) => ref
59     ↪ LinksIndexParts[node].LeftAsTarget;
60
61     /// <summary>
62     /// <para>
63     /// Gets the right reference using the specified node.
64     /// </para>
65     /// </summary>
66     /// <param name="node">
67     /// <para>The node.</para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLink GetRightReference(TLink node) => ref
74     ↪ LinksIndexParts[node].RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// </param>
84     /// <returns>
85     /// <para>The link</para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
89
90     /// <summary>
91     /// <para>
92     /// Gets the right using the specified node.
93     /// </para>
94     /// </summary>
95     /// <param name="node">
96     /// <para>The node.</para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;

```



```

109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↪ LinksIndexParts[node].LeftAsTarget = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↪ LinksIndexParts[node].RightAsTarget = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>
157    /// <para>The link</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
162
163    /// <summary>
164    /// <para>
165    /// Sets the size using the specified node.
166    /// </para>
167    /// <para></para>
168    /// </summary>
169    /// <param name="node">
170    /// <para>The node.</para>
171    /// <para></para>
172    /// </param>
173    /// <param name="size">
174    /// <para>The size.</para>
175    /// <para></para>
176    /// </param>
177    [MethodImpl(MethodImplOptions.AggressiveInlining)]
178    protected override void SetSize(TLink node, TLink size) =>
179        ↪ LinksIndexParts[node].SizeAsTarget = size;
180
181    /// <summary>
182    /// <para>
183    /// Gets the tree root using the specified node.
184    /// </para>
185    /// <para></para>
186    /// </summary>

```

```

184    /// <param name="node">
185    /// <para>The node.</para>
186    /// <para></para>
187    /// </param>
188    /// <returns>
189    /// <para>The link</para>
190    /// <para></para>
191    /// </returns>
192    [MethodImpl(MethodImplOptions.AggressiveInlining)]
193    protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsTarget;
194
195    /// <summary>
196    /// <para>
197    /// Gets the base part value using the specified node.
198    /// </para>
199    /// <para></para>
200    /// </summary>
201    /// <param name="node">
202    /// <para>The node.</para>
203    /// <para></para>
204    /// </param>
205    /// <returns>
206    /// <para>The link</para>
207    /// <para></para>
208    /// </returns>
209    [MethodImpl(MethodImplOptions.AggressiveInlining)]
210    protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
211
212    /// <summary>
213    /// <para>
214    /// Gets the key part value using the specified node.
215    /// </para>
216    /// <para></para>
217    /// </summary>
218    /// <param name="node">
219    /// <para>The node.</para>
220    /// <para></para>
221    /// </param>
222    /// <returns>
223    /// <para>The link</para>
224    /// <para></para>
225    /// </returns>
226    [MethodImpl(MethodImplOptions.AggressiveInlining)]
227    protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;
228
229    /// <summary>
230    /// <para>
231    /// Clears the node using the specified node.
232    /// </para>
233    /// <para></para>
234    /// </summary>
235    /// <param name="node">
236    /// <para>The node.</para>
237    /// <para></para>
238    /// </param>
239    [MethodImpl(MethodImplOptions.AggressiveInlining)]
240    protected override void ClearNode(TLink node)
241    {
242        ref var link = ref LinksIndexParts[node];
243        link.LeftAsTarget = Zero;
244        link.RightAsTarget = Zero;
245        link.SizeAsTarget = Zero;
246    }
247
248    /// <summary>
249    /// <para>
250    /// Searches the source.
251    /// </para>
252    /// <para></para>
253    /// </summary>
254    /// <param name="source">
255    /// <para>The source.</para>
256    /// <para></para>
257    /// </param>
258    /// <param name="target">
259    /// <para>The target.</para>
260    /// <para></para>
261    /// </param>

```

```

262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
        ↪ SearchCore(GetTreeRoot(target), source);
267 }
268 }

```

1.62 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethod

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksTargetsSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
        ↪ linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="node">
49         /// <para>The node.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>The ref link</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override ref TLink GetLeftReference(TLink node) => ref
        ↪ LinksIndexParts[node].LeftAsTarget;
58
59         /// <summary>
60         /// <para>
61         /// Gets the right reference using the specified node.
62         /// </para>

```

```

63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
75     ↪ LinksIndexParts[node].RightAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// <para></para>
120    /// </param>
121    /// <param name="left">
122    /// <para>The left.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected override void SetLeft(TLink node, TLink left) =>
127    ↪ LinksIndexParts[node].LeftAsTarget = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>

```

```

139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].RightAsTarget = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// </para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
159
160     /// <summary>
161     /// <para>
162     /// Sets the size using the specified node.
163     /// </para>
164     /// </summary>
165     /// <param name="node">
166     /// <para>The node.</para>
167     /// </para>
168     /// <param name="size">
169     /// <para>The size.</para>
170     /// </para>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetSize(TLink node, TLink size) =>
173         ↳ LinksIndexParts[node].SizeAsTarget = size;
174
175     /// <summary>
176     /// <para>
177     /// Gets the tree root using the specified node.
178     /// </para>
179     /// </summary>
180     /// <param name="node">
181     /// <para>The node.</para>
182     /// </para>
183     /// </param>
184     /// <returns>
185     /// <para>The link</para>
186     /// </returns>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsTarget;
189
190     /// <summary>
191     /// <para>
192     /// Gets the base part value using the specified node.
193     /// </para>
194     /// </summary>
195     /// <param name="node">
196     /// <para>The node.</para>
197     /// </para>
198     /// </param>
199     /// <returns>
200     /// <para>The link</para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
204
205     /// <summary>
206     /// <para>
207     /// Gets the key part value using the specified node.

```

```

215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="node">
219     /// <para>The node.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;
228
229     /// <summary>
230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref LinksIndexParts[node];
243         link.LeftAsTarget = Zero;
244         link.RightAsTarget = Zero;
245         link.SizeAsTarget = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(target), source);
268 }

```

1.63 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLink = System.UInt32;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 32 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
19     public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20     {
21         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;

```

```

23 private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24 private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25 private LinksHeader<TLink>* _header;
26 private RawLinkDataPart<TLink>* _linksDataParts;
27 private RawLinkIndexPart<TLink>* _linksIndexParts;
28
29 /// <summary>
30 /// <para>
31 /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
32 /// </para>
33 /// </summary>
34 /// <param name="dataMemory">
35 /// <para>A data memory.</para>
36 /// </param>
37 /// <param name="indexMemory">
38 /// <para>A index memory.</para>
39 /// </param>
40 [MethodImpl(MethodImplOptions.AggressiveInlining)]
41 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
42     ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
43
44 /// <summary>
45 /// <para>
46 /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
47 /// </para>
48 /// </summary>
49 /// <param name="dataMemory">
50 /// <para>A data memory.</para>
51 /// </param>
52 /// <param name="indexMemory">
53 /// <para>A index memory.</para>
54 /// </param>
55 /// <param name="memoryReservationStep">
56 /// <para>A memory reservation step.</para>
57 /// </param>
58 [MethodImpl(MethodImplOptions.AggressiveInlining)]
59 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
60     ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
61     ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
62     ↪ IndexTreeType.Default, useLinkedList: true) { }
63
64 /// <summary>
65 /// <para>
66 /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
67 /// </para>
68 /// </summary>
69 /// <param name="dataMemory">
70 /// <para>A data memory.</para>
71 /// </param>
72 /// <param name="indexMemory">
73 /// <para>A index memory.</para>
74 /// </param>
75 /// <param name="memoryReservationStep">
76 /// <para>A memory reservation step.</para>
77 /// </param>
78 /// <param name="constants">
79 /// <para>A constants.</para>
80 /// </param>
81 [MethodImpl(MethodImplOptions.AggressiveInlining)]
82 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
83     ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
84     ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
85     ↪ IndexTreeType.Default, useLinkedList: true) { }
86
87 /// <summary>
88 /// <para>

```

```

142  /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
143  /// </para>
144  /// </summary>
145  /// <param name="dataMemory">
146  /// <para>A data memory.</para>
147  /// </param>
148  /// <param name="indexMemory">
149  /// <para>A index memory.</para>
150  /// </param>
151  /// <param name="memoryReservationStep">
152  /// <para>A memory reservation step.</para>
153  /// </param>
154  /// <param name="constants">
155  /// <para>A constants.</para>
156  /// </param>
157  /// <param name="indexTreeType">
158  /// <para>A index tree type.</para>
159  /// </param>
160  /// <param name="useLinkedList">
161  /// <para>A use linked list.</para>
162  /// </param>
163  [MethodImpl(MethodImplOptions.AggressiveInlining)]
164  public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
165  ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
166  ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
167  ↪ memoryReservationStep, constants, useLinkedList)
168  {
169      if (indexTreeType == IndexTreeType.SizeBalancedTree)
170      {
171          _createInternalSourceTreeMethods = () => new
172          ↪ UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
173          ↪ _linksDataParts, _linksIndexParts, _header);
174          _createExternalSourceTreeMethods = () => new
175          ↪ UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
176          ↪ _linksDataParts, _linksIndexParts, _header);
177          _createInternalTargetTreeMethods = () => new
178          ↪ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
179          ↪ _linksDataParts, _linksIndexParts, _header);
180          _createExternalTargetTreeMethods = () => new
181          ↪ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
182          ↪ _linksDataParts, _linksIndexParts, _header);
183      }
184      else
185      {
186          _createInternalSourceTreeMethods = () => new
187          ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
188          ↪ _linksDataParts, _linksIndexParts, _header);
189          _createExternalSourceTreeMethods = () => new
190          ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
191          ↪ _linksDataParts, _linksIndexParts, _header);
192          _createInternalTargetTreeMethods = () => new
193          ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
194          ↪ _linksDataParts, _linksIndexParts, _header);
195          _createExternalTargetTreeMethods = () => new
196          ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
197          ↪ _linksDataParts, _linksIndexParts, _header);
198      }
199      Init(dataMemory, indexMemory);
200  }
201  /// <summary>
202  /// <para>
203  /// <para>Sets the pointers using the specified data memory.
204  /// </para>
205  /// </summary>
206  /// <param name="dataMemory">
207  /// <para>The data memory.</para>
208  /// </param>

```



```

152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// </para>
155 /// </param>
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 protected override void SetPointers(IResizableDirectMemory dataMemory,
158   ↳ IResizableDirectMemory indexMemory)
159 {
160     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
161     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
162     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
163     if (_useLinkedList)
164     {
165         InternalSourcesListMethods = new
166         ↳ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
167         ↳ _linksIndexParts);
168     }
169     else
170     {
171         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
172     }
173     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
174     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
175     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
176     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
177 }
178
179 /// <summary>
180 /// <para>
181 /// Resets the pointers.
182 /// </para>
183 /// </summary>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected override void ResetPointers()
186 {
187     base.ResetPointers();
188     _linksDataParts = null;
189     _linksIndexParts = null;
190     _header = null;
191 }
192
193 /// <summary>
194 /// <para>
195 /// Gets the header reference.
196 /// </para>
197 /// </summary>
198 /// <returns>
199 /// <para>A ref links header of t link</para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
203
204 /// <summary>
205 /// <para>
206 /// Gets the link data part reference using the specified link index.
207 /// </para>
208 /// </summary>
209 /// <param name="linkIndex">
210 /// <para>The link index.</para>
211 /// </param>
212 /// <returns>
213 /// <para>A ref raw link data part of t link</para>
214 /// </returns>
215 [MethodImpl(MethodImplOptions.AggressiveInlining)]
216 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
217   ↳ => ref _linksDataParts[linkIndex];
218
219 /// <summary>
220 /// <para>
221 /// Gets the link index part reference using the specified link index.
222 /// </para>
223 /// </summary>
224 /// <param name="linkIndex">
225 /// <para>The link index.</para>
226 /// </param>
227 /// <returns>
228 /// <para>A ref raw link index part of t link</para>
229 /// </returns>

```

```

226     /// </summary>
227     /// <param name="linkIndex">
228     /// <para>The link index.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>A ref raw link index part of t link</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
    ↪ linkIndex) => ref _linksIndexParts[linkIndex];

237     /// <summary>
238     /// <para>
239     /// <para>Determines whether this instance are equal.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="first">
244     /// <para>The first.</para>
245     /// <para></para>
246     /// </param>
247     /// <param name="second">
248     /// <para>The second.</para>
249     /// <para></para>
250     /// </param>
251     /// <returns>
252     /// <para>The bool</para>
253     /// <para></para>
254     /// </returns>
255     [MethodImpl(MethodImplOptions.AggressiveInlining)]
256     protected override bool AreEqual(TLink first, TLink second) => first == second;

257     /// <summary>
258     /// <para>
259     /// <para>Determines whether this instance less than.
260     /// </para>
261     /// <para></para>
262     /// </summary>
263     /// <param name="first">
264     /// <para>The first.</para>
265     /// <para></para>
266     /// </param>
267     /// <param name="second">
268     /// <para>The second.</para>
269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The bool</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override bool LessThan(TLink first, TLink second) => first < second;

277     /// <summary>
278     /// <para>
279     /// <para>Determines whether this instance less or equal than.
280     /// </para>
281     /// <para></para>
282     /// </summary>
283     /// <param name="first">
284     /// <para>The first.</para>
285     /// <para></para>
286     /// </param>
287     /// <param name="second">
288     /// <para>The second.</para>
289     /// <para></para>
290     /// </param>
291     /// <returns>
292     /// <para>The bool</para>
293     /// <para></para>
294     /// </returns>
295     [MethodImpl(MethodImplOptions.AggressiveInlining)]
296     protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;

297     /// <summary>
298     /// <para>

```

```

303     /// Determines whether this instance greater than.
304     /// </para>
305     /// <para></para>
306     /// </summary>
307     /// <param name="first">
308     /// <para>The first.</para>
309     /// <para></para>
310     /// </param>
311     /// <param name="second">
312     /// <para>The second.</para>
313     /// <para></para>
314     /// </param>
315     /// <returns>
316     /// <para>The bool</para>
317     /// <para></para>
318     /// </returns>
319     [MethodImpl(MethodImplOptions.AggressiveInlining)]
320     protected override bool GreaterThan(TLink first, TLink second) => first > second;
321
322     /// <summary>
323     /// <para>
324     /// Determines whether this instance greater or equal than.
325     /// </para>
326     /// <para></para>
327     /// </summary>
328     /// <param name="first">
329     /// <para>The first.</para>
330     /// <para></para>
331     /// </param>
332     /// <param name="second">
333     /// <para>The second.</para>
334     /// <para></para>
335     /// </param>
336     /// <returns>
337     /// <para>The bool</para>
338     /// <para></para>
339     /// </returns>
340     [MethodImpl(MethodImplOptions.AggressiveInlining)]
341     protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
342
343     /// <summary>
344     /// <para>
345     /// Gets the zero.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <returns>
350     /// <para>The link</para>
351     /// <para></para>
352     /// </returns>
353     [MethodImpl(MethodImplOptions.AggressiveInlining)]
354     protected override TLink GetZero() => 0U;
355
356     /// <summary>
357     /// <para>
358     /// Gets the one.
359     /// </para>
360     /// <para></para>
361     /// </summary>
362     /// <returns>
363     /// <para>The link</para>
364     /// <para></para>
365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override TLink GetOne() => 1U;
368
369     /// <summary>
370     /// <para>
371     /// Converts the to int 64 using the specified value.
372     /// </para>
373     /// <para></para>
374     /// </summary>
375     /// <param name="value">
376     /// <para>The value.</para>
377     /// <para></para>
378     /// </param>
379     /// <returns>
380     /// <para>The long</para>

```

```

381     /// <para></para>
382     /// </returns>
383     [MethodImpl(MethodImplOptions.AggressiveInlining)]
384     protected override long ConvertToInt64(TLink value) => value;
385
386     /// <summary>
387     /// <para>
388     /// Converts the to address using the specified value.
389     /// </para>
390     /// <para></para>
391     /// </summary>
392     /// <param name="value">
393     /// <para>The value.</para>
394     /// <para></para>
395     /// </param>
396     /// <returns>
397     /// <para>The link</para>
398     /// <para></para>
399     /// </returns>
400     [MethodImpl(MethodImplOptions.AggressiveInlining)]
401     protected override TLink ConvertToAddress(long value) => (TLink)value;
402
403     /// <summary>
404     /// <para>
405     /// Adds the first.
406     /// </para>
407     /// <para></para>
408     /// </summary>
409     /// <param name="first">
410     /// <para>The first.</para>
411     /// <para></para>
412     /// </param>
413     /// <param name="second">
414     /// <para>The second.</para>
415     /// <para></para>
416     /// </param>
417     /// <returns>
418     /// <para>The link</para>
419     /// <para></para>
420     /// </returns>
421     [MethodImpl(MethodImplOptions.AggressiveInlining)]
422     protected override TLink Add(TLink first, TLink second) => first + second;
423
424     /// <summary>
425     /// <para>
426     /// Subtracts the first.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The link</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override TLink Subtract(TLink first, TLink second) => first - second;
444
445     /// <summary>
446     /// <para>
447     /// Increments the link.
448     /// </para>
449     /// <para></para>
450     /// </summary>
451     /// <param name="link">
452     /// <para>The link.</para>
453     /// <para></para>
454     /// </param>
455     /// <returns>
456     /// <para>The link</para>
457     /// <para></para>
458     /// </returns>

```

```

459     [MethodImpl(MethodImplOptions.AggressiveInlining)]
460     protected override TLink Increment(TLink link) => ++link;
461
462     /// <summary>
463     /// <para>
464     /// Decrements the link.
465     /// </para>
466     /// <para></para>
467     /// </summary>
468     /// <param name="link">
469     /// <para>The link.</para>
470     /// <para></para>
471     /// </param>
472     /// <returns>
473     /// <para>The link</para>
474     /// <para></para>
475     /// </returns>
476     [MethodImpl(MethodImplOptions.AggressiveInlining)]
477     protected override TLink Decrement(TLink link) => --link;
478 }
479 }

```

1.64 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLink}"/>
16     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<TLink>
17     {
18         private readonly RawLinkDataPart<TLink>* _links;
19         private readonly LinksHeader<TLink>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
37             ↪ header)
38             : base((byte*)links, (byte*)header)
39         {
40             _links = links;
41             _header = header;
42         }
43
44         /// <summary>
45         /// <para>
46         /// Gets the link data part reference using the specified link.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="link">
51         /// <para>The link.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>A ref raw link data part of t link</para>
56         /// <para></para>

```

```

56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪ ref _links[link];
59
60     /// <summary>
61     /// <para>
62     /// Gets the header reference.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <returns>
67     /// <para>A ref links header of t link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
72 }
73 }

```

1.65 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTree

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 external links recursionless size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16     /// <seealso cref="ILinksTreeMethods{TLink}"/>
17     public unsafe abstract class UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
        ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The links data parts.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26         /// <summary>
27         /// <para>
28         /// The links index parts.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33         /// <summary>
34         /// <para>
35         /// The header.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected new readonly LinksHeader<TLink>* Header;
40
41         /// <summary>
42         /// <para>
43         /// Initializes a new <see
44         ↪ cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="constants">
49         /// <para>A constants.</para>
50         /// </param>
51         /// <param name="linksDataParts">
52         /// <para>A links data parts.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="linksIndexParts">

```

```

56  /// <para>A links index parts.</para>
57  /// <para></para>
58  /// </param>
59  /// <param name="header">
60  /// <para>A header.</para>
61  /// <para></para>
62  /// </param>
63  [MethodImpl(MethodImplOptions.AggressiveInlining)]
64  protected
    ↳ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header)
    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65  {
66  LinksDataParts = linksDataParts;
67  LinksIndexParts = linksIndexParts;
68  Header = header;
69  }
70
71  /// <summary>
72  /// <para>
73  /// Gets the zero.
74  /// </para>
75  /// <para></para>
76  /// </summary>
77  /// <returns>
78  /// <para>The ulong</para>
79  /// <para></para>
80  /// </returns>
81  [MethodImpl(MethodImplOptions.AggressiveInlining)]
82  protected override ulong GetZero() => OUL;
83
84  /// <summary>
85  /// <para>
86  /// Determines whether this instance equal to zero.
87  /// </para>
88  /// <para></para>
89  /// </summary>
90  /// <param name="value">
91  /// <para>The value.</para>
92  /// <para></para>
93  /// </param>
94  /// <returns>
95  /// <para>The bool</para>
96  /// <para></para>
97  /// </returns>
98  [MethodImpl(MethodImplOptions.AggressiveInlining)]
99  protected override bool EqualToZero(ulong value) => value == OUL;
100
101  /// <summary>
102  /// <para>
103  /// Determines whether this instance are equal.
104  /// </para>
105  /// <para></para>
106  /// </summary>
107  /// <param name="first">
108  /// <para>The first.</para>
109  /// <para></para>
110  /// </param>
111  /// <param name="second">
112  /// <para>The second.</para>
113  /// <para></para>
114  /// </param>
115  /// <returns>
116  /// <para>The bool</para>
117  /// <para></para>
118  /// </returns>
119  [MethodImpl(MethodImplOptions.AggressiveInlining)]
120  protected override bool AreEqual(ulong first, ulong second) => first == second;
121
122  /// <summary>
123  /// <para>
124  /// Determines whether this instance greater than zero.
125  /// </para>
126  /// <para></para>
127  /// </summary>
128  /// <param name="value">
129  /// <para>The value.</para>
130  /// <para></para>

```

```

131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The bool</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected override bool GreaterThanZero(ulong value) => value > 0UL;
139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance greater than.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="first">
147     /// <para>The first.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="second">
151     /// <para>The second.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The bool</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override bool GreaterThan(ulong first, ulong second) => first > second;
160
161     /// <summary>
162     /// <para>
163     /// Determines whether this instance greater or equal than.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="first">
168     /// <para>The first.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="second">
172     /// <para>The second.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
181
182     /// <summary>
183     /// <para>
184     /// Determines whether this instance greater or equal than zero.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="value">
189     /// <para>The value.</para>
190     /// <para></para>
191     /// </param>
192     /// <returns>
193     /// <para>The bool</para>
194     /// <para></para>
195     /// </returns>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
198     ↪ always true for ulong
199
200     /// <summary>
201     /// <para>
202     /// Determines whether this instance less or equal than zero.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="value">
207     /// <para>The value.</para>
208     /// <para></para>

```



```

208     /// </param>
209     /// <returns>
210     /// <para>The bool</para>
211     /// <para></para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
215
216     /// <summary>
217     /// <para>
218     /// Determines whether this instance less or equal than.
219     /// </para>
220     /// <para></para>
221     /// </summary>
222     /// <param name="first">
223     /// <para>The first.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="second">
227     /// <para>The second.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
236
237     /// <summary>
238     /// <para>
239     /// Determines whether this instance less than zero.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="value">
244     /// <para>The value.</para>
245     /// <para></para>
246     /// </param>
247     /// <returns>
248     /// <para>The bool</para>
249     /// <para></para>
250     /// </returns>
251     [MethodImpl(MethodImplOptions.AggressiveInlining)]
252     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong
253
254     /// <summary>
255     /// <para>
256     /// Determines whether this instance less than.
257     /// </para>
258     /// <para></para>
259     /// </summary>
260     /// <param name="first">
261     /// <para>The first.</para>
262     /// <para></para>
263     /// </param>
264     /// <param name="second">
265     /// <para>The second.</para>
266     /// <para></para>
267     /// </param>
268     /// <returns>
269     /// <para>The bool</para>
270     /// <para></para>
271     /// </returns>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override bool LessThan(ulong first, ulong second) => first < second;
274
275     /// <summary>
276     /// <para>
277     /// Increments the value.
278     /// </para>
279     /// <para></para>
280     /// </summary>
281     /// <param name="value">
282     /// <para>The value.</para>
283     /// <para></para>

```

```

284    /// </param>
285    /// <returns>
286    /// <para>The ulong</para>
287    /// <para></para>
288    /// </returns>
289    [MethodImpl(MethodImplOptions.AggressiveInlining)]
290    protected override ulong Increment(ulong value) => ++value;
291
292    /// <summary>
293    /// <para>
294    /// Decrements the value.
295    /// </para>
296    /// <para></para>
297    /// </summary>
298    /// <param name="value">
299    /// <para>The value.</para>
300    /// <para></para>
301    /// </param>
302    /// <returns>
303    /// <para>The ulong</para>
304    /// <para></para>
305    /// </returns>
306    [MethodImpl(MethodImplOptions.AggressiveInlining)]
307    protected override ulong Decrement(ulong value) => --value;
308
309    /// <summary>
310    /// <para>
311    /// Adds the first.
312    /// </para>
313    /// <para></para>
314    /// </summary>
315    /// <param name="first">
316    /// <para>The first.</para>
317    /// <para></para>
318    /// </param>
319    /// <param name="second">
320    /// <para>The second.</para>
321    /// <para></para>
322    /// </param>
323    /// <returns>
324    /// <para>The ulong</para>
325    /// <para></para>
326    /// </returns>
327    [MethodImpl(MethodImplOptions.AggressiveInlining)]
328    protected override ulong Add(ulong first, ulong second) => first + second;
329
330    /// <summary>
331    /// <para>
332    /// Subtracts the first.
333    /// </para>
334    /// <para></para>
335    /// </summary>
336    /// <param name="first">
337    /// <para>The first.</para>
338    /// <para></para>
339    /// </param>
340    /// <param name="second">
341    /// <para>The second.</para>
342    /// <para></para>
343    /// </param>
344    /// <returns>
345    /// <para>The ulong</para>
346    /// <para></para>
347    /// </returns>
348    [MethodImpl(MethodImplOptions.AggressiveInlining)]
349    protected override ulong Subtract(ulong first, ulong second) => first - second;
350
351    /// <summary>
352    /// <para>
353    /// Gets the header reference.
354    /// </para>
355    /// <para></para>
356    /// </summary>
357    /// <returns>
358    /// <para>A ref links header of t link</para>
359    /// <para></para>
360    /// </returns>
361    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

362     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364     /// <summary>
365     /// <para>
366     /// Gets the link data part reference using the specified link.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380         ↪ ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398         ↪ ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="first">
407     /// <para>The first.</para>
408     /// <para></para>
409     /// </param>
410     /// <param name="second">
411     /// <para>The second.</para>
412     /// <para></para>
413     /// </param>
414     /// <returns>
415     /// <para>The bool</para>
416     /// <para></para>
417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
420     {
421         ref var firstLink = ref LinksDataParts[first];
422         ref var secondLink = ref LinksDataParts[second];
423         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
424             ↪ secondLink.Source, secondLink.Target);
425     }
426
427     /// <summary>
428     /// <para>
429     /// Determines whether this instance first is to the right of second.
430     /// </para>
431     /// <para></para>
432     /// </summary>
433     /// <param name="first">
434     /// <para>The first.</para>
435     /// <para></para>
436     /// </param>
437     /// <param name="second">
438     /// <para>The second.</para>
439     /// <para></para>
440     /// </param>
441     /// <returns>
442     /// <para>The bool</para>
443     /// <para></para>
444     /// </returns>

```

```

437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
448             ↪ secondLink.Source, secondLink.Target);
449     }
450 }

```

1.66 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 external links size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16     /// <seealso cref="ILinksTreeMethods{TLink}"/>
17     public unsafe abstract class UInt64ExternalLinksSizeBalancedTreeMethodsBase :
18     ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// The links data parts.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
27         /// <summary>
28         /// <para>
29         /// The links index parts.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
34         /// <summary>
35         /// <para>
36         /// The header.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         protected new readonly LinksHeader<TLink>* Header;
41
42         /// <summary>
43         /// <para>
44         /// Initializes a new <see cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
45         ↪ instance.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="constants">
50         /// <para>A constants.</para>
51         /// <para></para>
52         /// </param>
53         /// <param name="linksDataParts">
54         /// <para>A links data parts.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="linksIndexParts">
58         /// <para>A links index parts.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="header">

```

```

60    /// <para>A header.</para>
61    /// <para></para>
62    /// </param>
63    [MethodImpl(MethodImplOptions.AggressiveInlining)]
64    protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↪ linksIndexParts, LinksHeader<TLink>* header)
    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65    {
66        LinksDataParts = linksDataParts;
67        LinksIndexParts = linksIndexParts;
68        Header = header;
69    }
70
71
72    /// <summary>
73    /// <para>
74    /// Gets the zero.
75    /// </para>
76    /// <para></para>
77    /// </summary>
78    /// <returns>
79    /// <para>The ulong</para>
80    /// <para></para>
81    /// </returns>
82    [MethodImpl(MethodImplOptions.AggressiveInlining)]
83    protected override ulong GetZero() => OUL;
84
85    /// <summary>
86    /// <para>
87    /// Determines whether this instance equal to zero.
88    /// </para>
89    /// <para></para>
90    /// </summary>
91    /// <param name="value">
92    /// <para>The value.</para>
93    /// <para></para>
94    /// </param>
95    /// <returns>
96    /// <para>The bool</para>
97    /// <para></para>
98    /// </returns>
99    [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(ulong value) => value == OUL;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// <para></para>
119    /// </returns>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override bool AreEqual(ulong first, ulong second) => first == second;
122
123    /// <summary>
124    /// <para>
125    /// Determines whether this instance greater than zero.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="value">
130    /// <para>The value.</para>
131    /// <para></para>
132    /// </param>
133    /// <returns>
134    /// <para>The bool</para>
135    /// <para></para>

```

```

136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected override bool GreaterThanZero(ulong value) => value > 0UL;
139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance greater than.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="first">
147     /// <para>The first.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="second">
151     /// <para>The second.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The bool</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override bool GreaterThan(ulong first, ulong second) => first > second;
160
161     /// <summary>
162     /// <para>
163     /// Determines whether this instance greater or equal than.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="first">
168     /// <para>The first.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="second">
172     /// <para>The second.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
181
182     /// <summary>
183     /// <para>
184     /// Determines whether this instance greater or equal than zero.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="value">
189     /// <para>The value.</para>
190     /// <para></para>
191     /// </param>
192     /// <returns>
193     /// <para>The bool</para>
194     /// <para></para>
195     /// </returns>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ always true for ulong
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance less or equal than zero.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="value">
206     /// <para>The value.</para>
207     /// <para></para>
208     /// </param>
209     /// <returns>
210     /// <para>The bool</para>
211     /// <para></para>
212     /// </returns>

```

```

213 [MethodImpl(MethodImplOptions.AggressiveInlining)]
214 protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↳ always >= 0 for ulong
215
216 /// <summary>
217 /// <para>
218 /// Determines whether this instance less or equal than.
219 /// </para>
220 /// <para></para>
221 /// </summary>
222 /// <param name="first">
223 /// <para>The first.</para>
224 /// <para></para>
225 /// </param>
226 /// <param name="second">
227 /// <para>The second.</para>
228 /// <para></para>
229 /// </param>
230 /// <returns>
231 /// <para>The bool</para>
232 /// <para></para>
233 /// </returns>
234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
236
237 /// <summary>
238 /// <para>
239 /// Determines whether this instance less than zero.
240 /// </para>
241 /// <para></para>
242 /// </summary>
243 /// <param name="value">
244 /// <para>The value.</para>
245 /// <para></para>
246 /// </param>
247 /// <returns>
248 /// <para>The bool</para>
249 /// <para></para>
250 /// </returns>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↳ for ulong
253
254 /// <summary>
255 /// <para>
256 /// Determines whether this instance less than.
257 /// </para>
258 /// <para></para>
259 /// </summary>
260 /// <param name="first">
261 /// <para>The first.</para>
262 /// <para></para>
263 /// </param>
264 /// <param name="second">
265 /// <para>The second.</para>
266 /// <para></para>
267 /// </param>
268 /// <returns>
269 /// <para>The bool</para>
270 /// <para></para>
271 /// </returns>
272 [MethodImpl(MethodImplOptions.AggressiveInlining)]
273 protected override bool LessThan(ulong first, ulong second) => first < second;
274
275 /// <summary>
276 /// <para>
277 /// Increments the value.
278 /// </para>
279 /// <para></para>
280 /// </summary>
281 /// <param name="value">
282 /// <para>The value.</para>
283 /// <para></para>
284 /// </param>
285 /// <returns>
286 /// <para>The ulong</para>
287 /// <para></para>
288 /// </returns>

```

```

289 [MethodImpl(MethodImplOptions.AggressiveInlining)]
290 protected override ulong Increment(ulong value) => ++value;
291
292 /// <summary>
293 /// <para>
294 /// Decrements the value.
295 /// </para>
296 /// <para></para>
297 /// </summary>
298 /// <param name="value">
299 /// <para>The value.</para>
300 /// <para></para>
301 /// </param>
302 /// <returns>
303 /// <para>The ulong</para>
304 /// <para></para>
305 /// </returns>
306 [MethodImpl(MethodImplOptions.AggressiveInlining)]
307 protected override ulong Decrement(ulong value) => --value;
308
309 /// <summary>
310 /// <para>
311 /// Adds the first.
312 /// </para>
313 /// <para></para>
314 /// </summary>
315 /// <param name="first">
316 /// <para>The first.</para>
317 /// <para></para>
318 /// </param>
319 /// <param name="second">
320 /// <para>The second.</para>
321 /// <para></para>
322 /// </param>
323 /// <returns>
324 /// <para>The ulong</para>
325 /// <para></para>
326 /// </returns>
327 [MethodImpl(MethodImplOptions.AggressiveInlining)]
328 protected override ulong Add(ulong first, ulong second) => first + second;
329
330 /// <summary>
331 /// <para>
332 /// Subtracts the first.
333 /// </para>
334 /// <para></para>
335 /// </summary>
336 /// <param name="first">
337 /// <para>The first.</para>
338 /// <para></para>
339 /// </param>
340 /// <param name="second">
341 /// <para>The second.</para>
342 /// <para></para>
343 /// </param>
344 /// <returns>
345 /// <para>The ulong</para>
346 /// <para></para>
347 /// </returns>
348 [MethodImpl(MethodImplOptions.AggressiveInlining)]
349 protected override ulong Subtract(ulong first, ulong second) => first - second;
350
351 /// <summary>
352 /// <para>
353 /// Gets the header reference.
354 /// </para>
355 /// <para></para>
356 /// </summary>
357 /// <returns>
358 /// <para>A ref links header of t link</para>
359 /// <para></para>
360 /// </returns>
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 protected override ref LinkHeader<TLink> GetHeaderReference() => ref *Header;
363
364 /// <summary>
365 /// <para>
366 /// Gets the link data part reference using the specified link.

```



```

367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380         ↪ ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398         ↪ ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="first">
407     /// <para>The first.</para>
408     /// <para></para>
409     /// </param>
410     /// <param name="second">
411     /// <para>The second.</para>
412     /// <para></para>
413     /// </param>
414     /// <returns>
415     /// <para>The bool</para>
416     /// <para></para>
417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
420     {
421         ref var firstLink = ref LinksDataParts[first];
422         ref var secondLink = ref LinksDataParts[second];
423         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
424             ↪ secondLink.Source, secondLink.Target);
425     }
426
427     /// <summary>
428     /// <para>
429     /// Determines whether this instance first is to the right of second.
430     /// </para>
431     /// <para></para>
432     /// </summary>
433     /// <param name="first">
434     /// <para>The first.</para>
435     /// <para></para>
436     /// </param>
437     /// <param name="second">
438     /// <para>The second.</para>
439     /// <para></para>
440     /// </param>
441     /// <returns>
442     /// <para>The bool</para>
443     /// <para></para>
444     /// </returns>

```

```

442 [MethodImpl(MethodImplOptions.AggressiveInlining)]
443 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
444 {
445     ref var firstLink = ref LinksDataParts[first];
446     ref var secondLink = ref LinksDataParts[second];
447     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
448         ↪ secondLink.Source, secondLink.Target);
449 }
450 }

```

1.67 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalanced

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 external links sources recursionless size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16    ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see
21        ↪ cref="UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public
43        ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46        ↪ linksIndexParts, header) { }
47
48        /// <summary>
49        /// <para>
50        /// Gets the left reference using the specified node.
51        /// </para>
52        /// <para></para>
53        /// </summary>
54        /// <param name="node">
55        /// <para>The node.</para>
56        /// <para></para>
57        /// </param>
58        /// <returns>
59        /// <para>The ref link</para>
60        /// <para></para>
61        /// </returns>
62        [MethodImpl(MethodImplOptions.AggressiveInlining)]
63        protected override ref TLink GetLeftReference(TLink node) => ref
64        ↪ LinksIndexParts[node].LeftAsSource;
65
66        /// <summary>

```

```

60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// <para></para>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// <para></para>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// <para></para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
    ↪ LinksIndexParts[node].LeftAsSource = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">
134    /// <para>The node.</para>
135    /// <para></para>

```

```

136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].RightAsSource = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="node">
152     /// <para>The node.</para>
153     /// <para></para>
154     /// </param>
155     /// <returns>
156     /// <para>The link</para>
157     /// <para></para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLink node, TLink size) =>
178         ↳ LinksIndexParts[node].SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLink GetTreeRoot() => Header->RootAsSource;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance first is to the left of second.
213     /// </para>

```

```

212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236     ↪ TLink secondSource, TLink secondTarget)
237     => firstSource < secondSource || firstSource == secondSource && firstTarget <
238     ↪ secondTarget;
239
240     /// <summary>
241     /// <para>
242     /// <para>Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268     ↪ TLink secondSource, TLink secondTarget)
269     => firstSource > secondSource || firstSource == secondSource && firstTarget >
270     ↪ secondTarget;
271
272     /// <summary>
273     /// <para>
274     /// <para>Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsSource = Zero;
287         link.RightAsSource = Zero;
288         link.SizeAsSource = Zero;
289     }

```

```
286     }
287 }
```

1.68 ./csharp/Platform.Data.Doublets.Memory.Split.Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods

```
1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 external links sources size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64ExternalLinksSourcesSizeBalancedTreeMethods :
16        ↳ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt64ExternalLinksSourcesSizeBalancedTreeMethods"/>
21        ↳ instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
43            ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44            ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45            ↳ linksIndexParts, header) { }
46
47        /// <summary>
48        /// <para>
49        /// Gets the left reference using the specified node.
50        /// </para>
51        /// <para></para>
52        /// </summary>
53        /// <param name="node">
54        /// <para>The node.</para>
55        /// <para></para>
56        /// </param>
57        /// <returns>
58        /// <para>The ref link</para>
59        /// <para></para>
60        /// </returns>
61        [MethodImpl(MethodImplOptions.AggressiveInlining)]
62        protected override ref TLink GetLeftReference(TLink node) => ref
63            ↳ LinksIndexParts[node].LeftAsSource;
64
65        /// <summary>
66        /// <para>
67        /// Gets the right reference using the specified node.
68        /// </para>
69        /// <para></para>
70        /// </summary>
71        /// <param name="node">
72        /// <para>The node.</para>
73        /// <para></para>
74        /// </param>
```

```

69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
75     ↪ LinksIndexParts[node].RightAsSource;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// <para></para>
120    /// </param>
121    /// <param name="left">
122    /// <para>The left.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected override void SetLeft(TLink node, TLink left) =>
127    ↪ LinksIndexParts[node].LeftAsSource = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>
141    /// <para></para>
142    /// </param>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void SetRight(TLink node, TLink right) =>
145    ↪ LinksIndexParts[node].RightAsSource = right;

```

```

144    /// <summary>
145    /// <para>
146    /// Gets the size using the specified node.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="node">
151    /// <para>The node.</para>
152    /// <para></para>
153    /// </param>
154    /// <returns>
155    /// <para>The link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
160
161    /// <summary>
162    /// <para>
163    /// Sets the size using the specified node.
164    /// </para>
165    /// <para></para>
166    /// </summary>
167    /// <param name="node">
168    /// <para>The node.</para>
169    /// <para></para>
170    /// </param>
171    /// <param name="size">
172    /// <para>The size.</para>
173    /// <para></para>
174    /// </param>
175    [MethodImpl(MethodImplOptions.AggressiveInlining)]
176    protected override void SetSize(TLink node, TLink size) =>
177        ↪ LinksIndexParts[node].SizeAsSource = size;
178
179    /// <summary>
180    /// <para>
181    /// Gets the tree root.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <returns>
186    /// <para>The link</para>
187    /// <para></para>
188    /// </returns>
189    [MethodImpl(MethodImplOptions.AggressiveInlining)]
190    protected override TLink GetTreeRoot() => Header->RootAsSource;
191
192    /// <summary>
193    /// <para>
194    /// Gets the base part value using the specified node.
195    /// </para>
196    /// <para></para>
197    /// </summary>
198    /// <param name="node">
199    /// <para>The node.</para>
200    /// <para></para>
201    /// </param>
202    /// <returns>
203    /// <para>The link</para>
204    /// <para></para>
205    /// </returns>
206    [MethodImpl(MethodImplOptions.AggressiveInlining)]
207    protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance first is to the left of second.
212    /// </para>
213    /// <para></para>
214    /// </summary>
215    /// <param name="firstSource">
216    /// <para>The first source.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="firstTarget">
220    /// <para>The first target.</para>
221    /// <para></para>

```



```

221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↪ TLink secondSource, TLink secondTarget)
237         => firstSource < secondSource || firstSource == secondSource && firstTarget <
238         ↪ secondTarget;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268         ↪ TLink secondSource, TLink secondTarget)
269         => firstSource > secondSource || firstSource == secondSource && firstTarget >
270         ↪ secondTarget;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsSource = Zero;
287         link.RightAsSource = Zero;
288         link.SizeAsSource = Zero;
289     }
290 }

```

1.69 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalance

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5

```

```

6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 external links targets recursionless size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16    ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see
21        ↪ cref="UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public
43        ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46        ↪ linksIndexParts, header) { }
47
48        /// <summary>
49        /// <para>
50        /// Gets the left reference using the specified node.
51        /// </para>
52        /// <para></para>
53        /// </summary>
54        /// <param name="node">
55        /// <para>The node.</para>
56        /// <para></para>
57        /// </param>
58        /// <returns>
59        /// <para>The ref link</para>
60        /// <para></para>
61        /// </returns>
62        [MethodImpl(MethodImplOptions.AggressiveInlining)]
63        protected override ref TLink GetLeftReference(TLink node) => ref
64        ↪ LinksIndexParts[node].LeftAsTarget;
65
66        /// <summary>
67        /// <para>
68        /// Gets the right reference using the specified node.
69        /// </para>
70        /// <para></para>
71        /// </summary>
72        /// <param name="node">
73        /// <para>The node.</para>
74        /// <para></para>
75        /// </param>
76        /// <returns>
77        /// <para>The ref link</para>
78        /// <para></para>
79        /// </returns>
80        [MethodImpl(MethodImplOptions.AggressiveInlining)]
81        protected override ref TLink GetRightReference(TLink node) => ref
82        ↪ LinksIndexParts[node].RightAsTarget;
83    }
84 }

```

```

76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↪ LinksIndexParts[node].LeftAsTarget = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↪ LinksIndexParts[node].RightAsTarget = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>

```

```

152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
160
161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLink node, TLink size) =>
177     ↪ LinksIndexParts[node].SizeAsTarget = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <returns>
186     /// <para>The link</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override TLink GetTreeRoot() => Header->RootAsTarget;
191
192     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified node.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
208
209     /// <summary>
210     /// <para>
211     /// Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>

```

```

229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↪ TLink secondSource, TLink secondTarget)
237         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
238             ↪ secondSource;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268         ↪ TLink secondSource, TLink secondTarget)
269         => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
270             ↪ secondSource;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsTarget = Zero;
287         link.RightAsTarget = Zero;
288         link.SizeAsTarget = Zero;
289     }
290 }

```

1.70 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 64 external links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>

```

```

14  /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
15  public unsafe class UInt64ExternalLinksTargetsSizeBalancedTreeMethods :
    ↳ UInt64ExternalLinksSizeBalancedTreeMethodsBase
16  {
17      /// <summary>
18      /// <para>
19      /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
    ↳ instance.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      /// <param name="constants">
24      /// <para>A constants.</para>
25      /// <para></para>
26      /// </param>
27      /// <param name="linksDataParts">
28      /// <para>A links data parts.</para>
29      /// <para></para>
30      /// </param>
31      /// <param name="linksIndexParts">
32      /// <para>A links index parts.</para>
33      /// <para></para>
34      /// </param>
35      /// <param name="header">
36      /// <para>A header.</para>
37      /// <para></para>
38      /// </param>
39      [MethodImpl(MethodImplOptions.AggressiveInlining)]
40      public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↳ linksIndexParts, header) { }
41
42      /// <summary>
43      /// <para>
44      /// Gets the left reference using the specified node.
45      /// </para>
46      /// <para></para>
47      /// </summary>
48      /// <param name="node">
49      /// <para>The node.</para>
50      /// <para></para>
51      /// </param>
52      /// <returns>
53      /// <para>The ref link</para>
54      /// <para></para>
55      /// </returns>
56      [MethodImpl(MethodImplOptions.AggressiveInlining)]
57      protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ LinksIndexParts[node].LeftAsTarget;
58
59      /// <summary>
60      /// <para>
61      /// Gets the right reference using the specified node.
62      /// </para>
63      /// <para></para>
64      /// </summary>
65      /// <param name="node">
66      /// <para>The node.</para>
67      /// <para></para>
68      /// </param>
69      /// <returns>
70      /// <para>The ref link</para>
71      /// <para></para>
72      /// </returns>
73      [MethodImpl(MethodImplOptions.AggressiveInlining)]
74      protected override ref TLink GetRightReference(TLink node) => ref
    ↳ LinksIndexParts[node].RightAsTarget;
75
76      /// <summary>
77      /// <para>
78      /// Gets the left using the specified node.
79      /// </para>
80      /// <para></para>
81      /// </summary>
82      /// <param name="node">
83      /// <para>The node.</para>
84      /// <para></para>

```

```

85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↪ LinksIndexParts[node].LeftAsTarget = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↪ LinksIndexParts[node].RightAsTarget = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>
157    /// <para>The link</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;

```

```

161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLink node, TLink size) =>
177         ↪ LinksIndexParts[node].SizeAsTarget = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <returns>
186     /// <para>The link</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override TLink GetTreeRoot() => Header->RootAsTarget;
191
192     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified node.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
208
209     /// <summary>
210     /// <para>
211     /// Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
237         ↪ TLink secondSource, TLink secondTarget)

```



```

236         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
           ↳ secondSource;
237
238     /// <summary>
239     /// <para>
240     /// Determines whether this instance first is to the right of second.
241     /// </para>
242     /// <para></para>
243     /// </summary>
244     /// <param name="firstSource">
245     /// <para>The first source.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="firstTarget">
249     /// <para>The first target.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondSource">
253     /// <para>The second source.</para>
254     /// <para></para>
255     /// </param>
256     /// <param name="secondTarget">
257     /// <para>The second target.</para>
258     /// <para></para>
259     /// </param>
260     /// <returns>
261     /// <para>The bool</para>
262     /// <para></para>
263     /// </returns>
264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
265     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
266         ↳ TLink secondSource, TLink secondTarget)
267         => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
268         ↳ secondSource;
269
270     /// <summary>
271     /// <para>
272     /// Clears the node using the specified node.
273     /// </para>
274     /// <para></para>
275     /// </summary>
276     /// <param name="node">
277     /// <para>The node.</para>
278     /// <para></para>
279     /// </param>
280     [MethodImpl(MethodImplOptions.AggressiveInlining)]
281     protected override void ClearNode(TLink node)
282     {
283         ref var link = ref LinksIndexParts[node];
284         link.LeftAsTarget = Zero;
285         link.RightAsTarget = Zero;
286         link.SizeAsTarget = Zero;
287     }
288 }

```

1.71 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeM

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt64;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 64 internal links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}">
16    public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17        ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>

```

```

22     /// <para></para>
23     /// </summary>
24     protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
25     /// <summary>
26     /// <para>
27     /// The links index parts.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
32     /// <summary>
33     /// <para>
34     /// The header.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected new readonly LinksHeader<TLink>* Header;
39
40     /// <summary>
41     /// <para>
42     /// Initializes a new <see
43     ↪ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// <para></para>
53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected
64     ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
65     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
66     ↪ linksIndexParts, LinksHeader<TLink>* header)
67     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68     {
69         LinksDataParts = linksDataParts;
70         LinksIndexParts = linksIndexParts;
71         Header = header;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Gets the zero.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetZero() => OUL;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance equal to zero.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="value">
94     /// <para>The value.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The bool</para>

```

```

96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(ulong value) => value == 0UL;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(ulong first, ulong second) => first == second;
121
122     /// <summary>
123     /// <para>
124     /// Determines whether this instance greater than zero.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The bool</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override bool GreaterThanZero(ulong value) => value > 0UL;
138
139     /// <summary>
140     /// <para>
141     /// Determines whether this instance greater than.
142     /// </para>
143     /// <para></para>
144     /// </summary>
145     /// <param name="first">
146     /// <para>The first.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="second">
150     /// <para>The second.</para>
151     /// <para></para>
152     /// </param>
153     /// <returns>
154     /// <para>The bool</para>
155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override bool GreaterThan(ulong first, ulong second) => first > second;
159
160     /// <summary>
161     /// <para>
162     /// Determines whether this instance greater or equal than.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="first">
167     /// <para>The first.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="second">
171     /// <para>The second.</para>
172     /// <para></para>
173     /// </param>

```

```

174    /// <returns>
175    /// <para>The bool</para>
176    /// <para></para>
177    /// </returns>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
180
181    /// <summary>
182    /// <para>
183    /// Determines whether this instance greater or equal than zero.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <param name="value">
188    /// <para>The value.</para>
189    /// <para></para>
190    /// </param>
191    /// <returns>
192    /// <para>The bool</para>
193    /// <para></para>
194    /// </returns>
195    [MethodImpl(MethodImplOptions.AggressiveInlining)]
196    protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ always true for ulong
197
198    /// <summary>
199    /// <para>
200    /// Determines whether this instance less or equal than zero.
201    /// </para>
202    /// <para></para>
203    /// </summary>
204    /// <param name="value">
205    /// <para>The value.</para>
206    /// <para></para>
207    /// </param>
208    /// <returns>
209    /// <para>The bool</para>
210    /// <para></para>
211    /// </returns>
212    [MethodImpl(MethodImplOptions.AggressiveInlining)]
213    protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
214
215    /// <summary>
216    /// <para>
217    /// Determines whether this instance less or equal than.
218    /// </para>
219    /// <para></para>
220    /// </summary>
221    /// <param name="first">
222    /// <para>The first.</para>
223    /// <para></para>
224    /// </param>
225    /// <param name="second">
226    /// <para>The second.</para>
227    /// <para></para>
228    /// </param>
229    /// <returns>
230    /// <para>The bool</para>
231    /// <para></para>
232    /// </returns>
233    [MethodImpl(MethodImplOptions.AggressiveInlining)]
234    protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
235
236    /// <summary>
237    /// <para>
238    /// Determines whether this instance less than zero.
239    /// </para>
240    /// <para></para>
241    /// </summary>
242    /// <param name="value">
243    /// <para>The value.</para>
244    /// <para></para>
245    /// </param>
246    /// <returns>
247    /// <para>The bool</para>
248    /// <para></para>
249    /// </returns>

```

```

250 [MethodImpl(MethodImplOptions.AggressiveInlining)]
251 protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↳ for ulong
252
253 /// <summary>
254 /// <para>
255 /// Determines whether this instance less than.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="first">
260 /// <para>The first.</para>
261 /// <para></para>
262 /// </param>
263 /// <param name="second">
264 /// <para>The second.</para>
265 /// <para></para>
266 /// </param>
267 /// <returns>
268 /// <para>The bool</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 protected override bool LessThan(ulong first, ulong second) => first < second;
273
274 /// <summary>
275 /// <para>
276 /// Increments the value.
277 /// </para>
278 /// <para></para>
279 /// </summary>
280 /// <param name="value">
281 /// <para>The value.</para>
282 /// <para></para>
283 /// </param>
284 /// <returns>
285 /// <para>The ulong</para>
286 /// <para></para>
287 /// </returns>
288 [MethodImpl(MethodImplOptions.AggressiveInlining)]
289 protected override ulong Increment(ulong value) => ++value;
290
291 /// <summary>
292 /// <para>
293 /// Decrements the value.
294 /// </para>
295 /// <para></para>
296 /// </summary>
297 /// <param name="value">
298 /// <para>The value.</para>
299 /// <para></para>
300 /// </param>
301 /// <returns>
302 /// <para>The ulong</para>
303 /// <para></para>
304 /// </returns>
305 [MethodImpl(MethodImplOptions.AggressiveInlining)]
306 protected override ulong Decrement(ulong value) => --value;
307
308 /// <summary>
309 /// <para>
310 /// Adds the first.
311 /// </para>
312 /// <para></para>
313 /// </summary>
314 /// <param name="first">
315 /// <para>The first.</para>
316 /// <para></para>
317 /// </param>
318 /// <param name="second">
319 /// <para>The second.</para>
320 /// <para></para>
321 /// </param>
322 /// <returns>
323 /// <para>The ulong</para>
324 /// <para></para>
325 /// </returns>
326 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

327     protected override ulong Add(ulong first, ulong second) => first + second;
328
329     /// <summary>
330     /// <para>
331     /// Subtracts the first.
332     /// </para>
333     /// <para></para>
334     /// </summary>
335     /// <param name="first">
336     /// <para>The first.</para>
337     /// <para></para>
338     /// </param>
339     /// <param name="second">
340     /// <para>The second.</para>
341     /// <para></para>
342     /// </param>
343     /// <returns>
344     /// <para>The ulong</para>
345     /// <para></para>
346     /// </returns>
347     [MethodImpl(MethodImplOptions.AggressiveInlining)]
348     protected override ulong Subtract(ulong first, ulong second) => first - second;
349
350     /// <summary>
351     /// <para>
352     /// Gets the link data part reference using the specified link.
353     /// </para>
354     /// <para></para>
355     /// </summary>
356     /// <param name="link">
357     /// <para>The link.</para>
358     /// <para></para>
359     /// </param>
360     /// <returns>
361     /// <para>A ref raw link data part of t link</para>
362     /// <para></para>
363     /// </returns>
364     [MethodImpl(MethodImplOptions.AggressiveInlining)]
365     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
366         ↪ ref LinksDataParts[link];
367
368     /// <summary>
369     /// <para>
370     /// Gets the link index part reference using the specified link.
371     /// </para>
372     /// <para></para>
373     /// </summary>
374     /// <param name="link">
375     /// <para>The link.</para>
376     /// <para></para>
377     /// </param>
378     /// <returns>
379     /// <para>A ref raw link index part of t link</para>
380     /// <para></para>
381     /// </returns>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
384         ↪ ref LinksIndexParts[link];
385
386     /// <summary>
387     /// <para>
388     /// Determines whether this instance first is to the left of second.
389     /// </para>
390     /// <para></para>
391     /// </summary>
392     /// <param name="first">
393     /// <para>The first.</para>
394     /// <para></para>
395     /// </param>
396     /// <param name="second">
397     /// <para>The second.</para>
398     /// <para></para>
399     /// </param>
400     /// <returns>
401     /// <para>The bool</para>
402     /// <para></para>
403     /// </returns>
404     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

403     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
404         ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
405
406     /// <summary>
407     /// <para>
408     /// Determines whether this instance first is to the right of second.
409     /// </para>
410     /// <para></para>
411     /// </summary>
412     /// <param name="first">
413     /// <para>The first.</para>
414     /// <para></para>
415     /// </param>
416     /// <param name="second">
417     /// <para>The second.</para>
418     /// <para></para>
419     /// </param>
420     /// <returns>
421     /// <para>The bool</para>
422     /// <para></para>
423     /// </returns>
424     [MethodImpl(MethodImplOptions.AggressiveInlining)]
425     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
426         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
427 }
428 }

```

1.72 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 internal links size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16     public unsafe abstract class UInt64InternalLinksSizeBalancedTreeMethodsBase :
17         ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The links data parts.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26         /// <summary>
27         /// <para>
28         /// The links index parts.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33         /// <summary>
34         /// <para>
35         /// The header.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected new readonly LinksHeader<TLink>* Header;
40
41         /// <summary>
42         /// <para>
43         /// Initializes a new <see cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
44         /// ↪ instance.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="constants">
49         /// <para>A constants.</para>
50         /// <para></para>

```

```

49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// <para></para>
53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected UInt64 InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↳ linksIndexParts, LinksHeader<TLink>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64     {
65     }
66     LinksDataParts = linksDataParts;
67     LinksIndexParts = linksIndexParts;
68     Header = header;
69 }
70
71     /// <summary>
72     /// <para>
73     /// Gets the zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>The ulong</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override ulong GetZero() => 0UL;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(ulong value) => value == 0UL;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(ulong first, ulong second) => first == second;
121
122     /// <summary>
123     /// <para>
124     /// Determines whether this instance greater than zero.

```



```

125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The bool</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override bool GreaterThanZero(ulong value) => value > 0UL;
138
139     /// <summary>
140     /// <para>
141     /// Determines whether this instance greater than.
142     /// </para>
143     /// <para></para>
144     /// </summary>
145     /// <param name="first">
146     /// <para>The first.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="second">
150     /// <para>The second.</para>
151     /// <para></para>
152     /// </param>
153     /// <returns>
154     /// <para>The bool</para>
155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override bool GreaterThan(ulong first, ulong second) => first > second;
159
160     /// <summary>
161     /// <para>
162     /// Determines whether this instance greater or equal than.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="first">
167     /// <para>The first.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="second">
171     /// <para>The second.</para>
172     /// <para></para>
173     /// </param>
174     /// <returns>
175     /// <para>The bool</para>
176     /// <para></para>
177     /// </returns>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance greater or equal than zero.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="value">
188     /// <para>The value.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The bool</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
197     ↪ always true for ulong
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance less or equal than zero.
202     /// </para>

```

```

202     /// <para></para>
203     /// </summary>
204     /// <param name="value">
205     /// <para>The value.</para>
206     /// <para></para>
207     /// </param>
208     /// <returns>
209     /// <para>The bool</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
214
215     /// <summary>
216     /// <para>
217     /// Determines whether this instance less or equal than.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="first">
222     /// <para>The first.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="second">
226     /// <para>The second.</para>
227     /// <para></para>
228     /// </param>
229     /// <returns>
230     /// <para>The bool</para>
231     /// <para></para>
232     /// </returns>
233     [MethodImpl(MethodImplOptions.AggressiveInlining)]
234     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance less than zero.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="value">
243     /// <para>The value.</para>
244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>The bool</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong
252
253     /// <summary>
254     /// <para>
255     /// Determines whether this instance less than.
256     /// </para>
257     /// <para></para>
258     /// </summary>
259     /// <param name="first">
260     /// <para>The first.</para>
261     /// <para></para>
262     /// </param>
263     /// <param name="second">
264     /// <para>The second.</para>
265     /// <para></para>
266     /// </param>
267     /// <returns>
268     /// <para>The bool</para>
269     /// <para></para>
270     /// </returns>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     protected override bool LessThan(ulong first, ulong second) => first < second;
273
274     /// <summary>
275     /// <para>
276     /// Increments the value.
277     /// </para>

```

```

278     /// <para></para>
279     /// </summary>
280     /// <param name="value">
281     /// <para>The value.</para>
282     /// <para></para>
283     /// </param>
284     /// <returns>
285     /// <para>The ulong</para>
286     /// <para></para>
287     /// </returns>
288     [MethodImpl(MethodImplOptions.AggressiveInlining)]
289     protected override ulong Increment(ulong value) => ++value;
290
291     /// <summary>
292     /// <para>
293     /// Decrements the value.
294     /// </para>
295     /// <para></para>
296     /// </summary>
297     /// <param name="value">
298     /// <para>The value.</para>
299     /// <para></para>
300     /// </param>
301     /// <returns>
302     /// <para>The ulong</para>
303     /// <para></para>
304     /// </returns>
305     [MethodImpl(MethodImplOptions.AggressiveInlining)]
306     protected override ulong Decrement(ulong value) => --value;
307
308     /// <summary>
309     /// <para>
310     /// Adds the first.
311     /// </para>
312     /// <para></para>
313     /// </summary>
314     /// <param name="first">
315     /// <para>The first.</para>
316     /// <para></para>
317     /// </param>
318     /// <param name="second">
319     /// <para>The second.</para>
320     /// <para></para>
321     /// </param>
322     /// <returns>
323     /// <para>The ulong</para>
324     /// <para></para>
325     /// </returns>
326     [MethodImpl(MethodImplOptions.AggressiveInlining)]
327     protected override ulong Add(ulong first, ulong second) => first + second;
328
329     /// <summary>
330     /// <para>
331     /// Subtracts the first.
332     /// </para>
333     /// <para></para>
334     /// </summary>
335     /// <param name="first">
336     /// <para>The first.</para>
337     /// <para></para>
338     /// </param>
339     /// <param name="second">
340     /// <para>The second.</para>
341     /// <para></para>
342     /// </param>
343     /// <returns>
344     /// <para>The ulong</para>
345     /// <para></para>
346     /// </returns>
347     [MethodImpl(MethodImplOptions.AggressiveInlining)]
348     protected override ulong Subtract(ulong first, ulong second) => first - second;
349
350     /// <summary>
351     /// <para>
352     /// Gets the link data part reference using the specified link.
353     /// </para>
354     /// <para></para>
355     /// </summary>

```

```

356     /// <param name="link">
357     /// <para>The link.</para>
358     /// <para></para>
359     /// </param>
360     /// <returns>
361     /// <para>A ref raw link data part of t link</para>
362     /// <para></para>
363     /// </returns>
364     [MethodImpl(MethodImplOptions.AggressiveInlining)]
365     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪ ref LinksDataParts[link];
366
367     /// <summary>
368     /// <para>
369     /// Gets the link index part reference using the specified link.
370     /// </para>
371     /// <para></para>
372     /// </summary>
373     /// <param name="link">
374     /// <para>The link.</para>
375     /// <para></para>
376     /// </param>
377     /// <returns>
378     /// <para>A ref raw link index part of t link</para>
379     /// <para></para>
380     /// </returns>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref LinksIndexParts[link];
383
384     /// <summary>
385     /// <para>
386     /// Determines whether this instance first is to the left of second.
387     /// </para>
388     /// <para></para>
389     /// </summary>
390     /// <param name="first">
391     /// <para>The first.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="second">
395     /// <para>The second.</para>
396     /// <para></para>
397     /// </param>
398     /// <returns>
399     /// <para>The bool</para>
400     /// <para></para>
401     /// </returns>
402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
403     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
        ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
404
405     /// <summary>
406     /// <para>
407     /// Determines whether this instance first is to the right of second.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="first">
412     /// <para>The first.</para>
413     /// <para></para>
414     /// </param>
415     /// <param name="second">
416     /// <para>The second.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
        ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

1.73 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Generic
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links sources linked list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}"/>
15     public unsafe class UInt64InternalLinksSourcesLinkedListMethods :
16     ↪ InternalLinksSourcesLinkedListMethods<TLink>
17     {
18         private readonly RawLinkDataPart<TLink>* _linksDataParts;
19         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods"/> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="constants">
28         /// <para>A constants.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksDataParts">
32         /// <para>A links data parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="linksIndexParts">
36         /// <para>A links index parts.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
41         ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
42         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
43         {
44             _linksDataParts = linksDataParts;
45             _linksIndexParts = linksIndexParts;
46         }
47
48         /// <summary>
49         /// <para>
50         /// Gets the link data part reference using the specified link.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="link">
55         /// <para>The link.</para>
56         /// <para></para>
57         /// </param>
58         /// <returns>
59         /// <para>A ref raw link data part of t link</para>
60         /// <para></para>
61         /// </returns>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
64         ↪ ref _linksDataParts[link];
65
66         /// <summary>
67         /// <para>
68         /// Gets the link index part reference using the specified link.
69         /// </para>
70         /// <para></para>
71         /// </summary>
72         /// <param name="link">
73         /// <para>The link.</para>
74         /// <para></para>
75         /// </param>
76         /// <returns>
77         /// <para>A ref raw link index part of t link</para>
78         /// <para></para>
79         /// </returns>
80     }
81 }

```

```

75     /// <para></para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
79         ↪ ref _linksIndexParts[link];
80 }

```

1.74 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16         ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public
43         ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46         ↪ linksIndexParts, header) { }
47
48         /// <summary>
49         /// <para>
50         /// Gets the left reference using the specified node.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="node">
55         /// <para>The node.</para>
56         /// <para></para>
57         /// </param>
58         /// <returns>
59         /// <para>The ref link</para>
60         /// <para></para>
61         /// </returns>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         protected override ref TLink GetLeftReference(TLink node) => ref
64         ↪ LinksIndexParts[node].LeftAsSource;
65
66         /// <summary>
67         /// <para>
68         /// Gets the right reference using the specified node.
69         /// </para>
70         /// <para></para>
71         /// </summary>
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         protected override ref TLink GetRightReference(TLink node) => ref
74         ↪ LinksIndexParts[node].RightAsSource;
75     }
76 }

```

```

63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
75     ↪ LinksIndexParts[node].RightAsSource;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// <para></para>
120    /// </param>
121    /// <param name="left">
122    /// <para>The left.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected override void SetLeft(TLink node, TLink left) =>
127    ↪ LinksIndexParts[node].LeftAsSource = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>

```

```

139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].RightAsSource = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// </para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
159
160     /// <summary>
161     /// <para>
162     /// Sets the size using the specified node.
163     /// </para>
164     /// </summary>
165     /// <param name="node">
166     /// <para>The node.</para>
167     /// </para>
168     /// <param name="size">
169     /// <para>The size.</para>
170     /// </para>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetSize(TLink node, TLink size) =>
173         ↳ LinksIndexParts[node].SizeAsSource = size;
174
175     /// <summary>
176     /// <para>
177     /// Gets the tree root using the specified node.
178     /// </para>
179     /// </summary>
180     /// <param name="node">
181     /// <para>The node.</para>
182     /// </para>
183     /// </param>
184     /// <returns>
185     /// <para>The link</para>
186     /// </returns>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsSource;
189
190     /// <summary>
191     /// <para>
192     /// Gets the base part value using the specified node.
193     /// </para>
194     /// </summary>
195     /// <param name="node">
196     /// <para>The node.</para>
197     /// </para>
198     /// </param>
199     /// <returns>
200     /// <para>The link</para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
204
205     /// <summary>
206     /// <para>
207     /// Gets the key part value using the specified node.

```



```

215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="node">
219     /// <para>The node.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;
228
229     /// <summary>
230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref LinksIndexParts[node];
243         link.LeftAsSource = Zero;
244         link.RightAsSource = Zero;
245         link.SizeAsSource = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(source), target);
268 }

```

1.75 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 64 internal links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksSourcesSizeBalancedTreeMethods :
16         ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64InternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>

```

```

21    /// <para></para>
22    /// </summary>
23    /// <param name="constants">
24    /// <para>A constants.</para>
25    /// <para></para>
26    /// </param>
27    /// <param name="linksDataParts">
28    /// <para>A links data parts.</para>
29    /// <para></para>
30    /// </param>
31    /// <param name="linksIndexParts">
32    /// <para>A links index parts.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt64InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↪ linksIndexParts, header) { }
41
42    /// <summary>
43    /// <para>
44    /// Gets the left reference using the specified node.
45    /// </para>
46    /// <para></para>
47    /// </summary>
48    /// <param name="node">
49    /// <para>The node.</para>
50    /// <para></para>
51    /// </param>
52    /// <returns>
53    /// <para>The ref link</para>
54    /// <para></para>
55    /// </returns>
56    [MethodImpl(MethodImplOptions.AggressiveInlining)]
57    protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;
58
59    /// <summary>
60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// <para></para>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// <para></para>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// <para></para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93    /// <summary>

```

```

94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↪ LinksIndexParts[node].LeftAsSource = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↪ LinksIndexParts[node].RightAsSource = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>
157    /// <para>The link</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
162
163    /// <summary>
164    /// <para>
165    /// Sets the size using the specified node.
166    /// </para>
167    /// <para></para>
168    /// </summary>
169    /// <param name="node">
170    /// <para>The node.</para>
171    /// <para></para>

```

```

170    /// </param>
171    /// <param name="size">
172    /// <para>The size.</para>
173    /// <para></para>
174    /// </param>
175    [MethodImpl(MethodImplOptions.AggressiveInlining)]
176    protected override void SetSize(TLink node, TLink size) =>
177        ↳ LinksIndexParts[node].SizeAsSource = size;
178
179    /// <summary>
180    /// <para>
181    /// Gets the tree root using the specified node.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <param name="node">
186    /// <para>The node.</para>
187    /// <para></para>
188    /// </param>
189    /// <returns>
190    /// <para>The link</para>
191    /// <para></para>
192    /// </returns>
193    [MethodImpl(MethodImplOptions.AggressiveInlining)]
194    protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsSource;
195
196    /// <summary>
197    /// <para>
198    /// Gets the base part value using the specified node.
199    /// </para>
200    /// <para></para>
201    /// </summary>
202    /// <param name="node">
203    /// <para>The node.</para>
204    /// <para></para>
205    /// </param>
206    /// <returns>
207    /// <para>The link</para>
208    /// <para></para>
209    /// </returns>
210    [MethodImpl(MethodImplOptions.AggressiveInlining)]
211    protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
212
213    /// <summary>
214    /// <para>
215    /// Gets the key part value using the specified node.
216    /// </para>
217    /// <para></para>
218    /// </summary>
219    /// <param name="node">
220    /// <para>The node.</para>
221    /// <para></para>
222    /// </param>
223    /// <returns>
224    /// <para>The link</para>
225    /// <para></para>
226    /// </returns>
227    [MethodImpl(MethodImplOptions.AggressiveInlining)]
228    protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;
229
230    /// <summary>
231    /// <para>
232    /// Clears the node using the specified node.
233    /// </para>
234    /// <para></para>
235    /// </summary>
236    /// <param name="node">
237    /// <para>The node.</para>
238    /// <para></para>
239    /// </param>
240    [MethodImpl(MethodImplOptions.AggressiveInlining)]
241    protected override void ClearNode(TLink node)
242    {
243        ref var link = ref LinksIndexParts[node];
244        link.LeftAsSource = Zero;
245        link.RightAsSource = Zero;
246        link.SizeAsSource = Zero;
247    }

```

```

247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
        ↪ SearchCore(GetTreeRoot(source), target);
267     }
268 }

```

1.76 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
        ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public
42         ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
43         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45         ↪ linksIndexParts, header) { }
46
47         /// <summary>
48         /// <para>
49         /// Gets the left reference using the specified node.
50         /// </para>
51         /// <para></para>
52         /// </summary>

```

```

48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref ulong</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ LinksIndexParts[node].LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref ulong</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref ulong GetRightReference(ulong node) => ref
    ↪ LinksIndexParts[node].RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>

```

```

124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 protected override void SetLeft(TLink node, TLink left) =>
    ↳ LinksIndexParts[node].LeftAsTarget = left;

126
127 /// <summary>
128 /// <para>
129 /// Sets the right using the specified node.
130 /// </para>
131 /// <para></para>
132 /// </summary>
133 /// <param name="node">
134 /// <para>The node.</para>
135 /// <para></para>
136 /// </param>
137 /// <param name="right">
138 /// <para>The right.</para>
139 /// <para></para>
140 /// </param>
141 [MethodImpl(MethodImplOptions.AggressiveInlining)]
142 protected override void SetRight(TLink node, TLink right) =>
    ↳ LinksIndexParts[node].RightAsTarget = right;

143
144 /// <summary>
145 /// <para>
146 /// Gets the size using the specified node.
147 /// </para>
148 /// <para></para>
149 /// </summary>
150 /// <param name="node">
151 /// <para>The node.</para>
152 /// <para></para>
153 /// </param>
154 /// <returns>
155 /// <para>The link</para>
156 /// <para></para>
157 /// </returns>
158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
160
161 /// <summary>
162 /// <para>
163 /// Sets the size using the specified node.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 /// <param name="node">
168 /// <para>The node.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="size">
172 /// <para>The size.</para>
173 /// <para></para>
174 /// </param>
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 protected override void SetSize(TLink node, TLink size) =>
    ↳ LinksIndexParts[node].SizeAsTarget = size;

177
178 /// <summary>
179 /// <para>
180 /// Gets the tree root using the specified node.
181 /// </para>
182 /// <para></para>
183 /// </summary>
184 /// <param name="node">
185 /// <para>The node.</para>
186 /// <para></para>
187 /// </param>
188 /// <returns>
189 /// <para>The link</para>
190 /// <para></para>
191 /// </returns>
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsTarget;
194
195 /// <summary>
196 /// <para>
197 /// Gets the base part value using the specified node.
198 /// </para>

```

```

199     /// <para></para>
200     /// </summary>
201     /// <param name="node">
202     /// <para>The node.</para>
203     /// <para></para>
204     /// </param>
205     /// <returns>
206     /// <para>The link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
211
212     /// <summary>
213     /// <para>
214     /// Gets the key part value using the specified node.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="node">
219     /// <para>The node.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;
228
229     /// <summary>
230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref LinksIndexParts[node];
243         link.LeftAsTarget = Zero;
244         link.RightAsTarget = Zero;
245         link.SizeAsTarget = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(target), source);
268 }

```

1.77 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethod

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5

```



```

6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 internal links targets size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64InternalLinksTargetsSizeBalancedTreeMethods :
16    ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
21        ↪ instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// </param>
31        /// <param name="linksIndexParts">
32        /// <para>A links index parts.</para>
33        /// </param>
34        /// <param name="header">
35        /// <para>A header.</para>
36        /// </param>
37        [MethodImpl(MethodImplOptions.AggressiveInlining)]
38        public UInt64InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
39        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
40        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
41        ↪ linksIndexParts, header) { }
42
43        /// <summary>
44        /// <para>
45        /// Gets the left reference using the specified node.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        /// <param name="node">
50        /// <para>The node.</para>
51        /// </param>
52        /// <returns>
53        /// <para>The ref ulong</para>
54        /// <para></para>
55        /// </returns>
56        [MethodImpl(MethodImplOptions.AggressiveInlining)]
57        protected override ref ulong GetLeftReference(ulong node) => ref
58        ↪ LinksIndexParts[node].LeftAsTarget;
59
60        /// <summary>
61        /// <para>
62        /// Gets the right reference using the specified node.
63        /// </para>
64        /// <para></para>
65        /// </summary>
66        /// <param name="node">
67        /// <para>The node.</para>
68        /// </param>
69        /// <returns>
70        /// <para>The ref ulong</para>
71        /// <para></para>
72        /// </returns>
73        [MethodImpl(MethodImplOptions.AggressiveInlining)]
74        protected override ref ulong GetRightReference(ulong node) => ref
75        ↪ LinksIndexParts[node].RightAsTarget;
76
77        /// <summary>

```

```

77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
126        ↪ LinksIndexParts[node].LeftAsTarget = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLink node, TLink right) =>
144        ↪ LinksIndexParts[node].RightAsTarget = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>

```

```

153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
160
161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLink node, TLink size) =>
177     ↪ LinksIndexParts[node].SizeAsTarget = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root using the specified node.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="node">
186     /// <para>The node.</para>
187     /// <para></para>
188     /// </param>
189     /// <returns>
190     /// <para>The link</para>
191     /// <para></para>
192     /// </returns>
193     [MethodImpl(MethodImplOptions.AggressiveInlining)]
194     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsTarget;
195
196     /// <summary>
197     /// <para>
198     /// Gets the base part value using the specified node.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     /// <param name="node">
203     /// <para>The node.</para>
204     /// <para></para>
205     /// </param>
206     /// <returns>
207     /// <para>The link</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
212
213     /// <summary>
214     /// <para>
215     /// Gets the key part value using the specified node.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="node">
220     /// <para>The node.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The link</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;
229
230     /// <summary>

```

```

230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLink node)
241     {
242         ref var link = ref LinksIndexParts[node];
243         link.LeftAsTarget = Zero;
244         link.RightAsTarget = Zero;
245         link.SizeAsTarget = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(target), source);
268 }

```

1.78 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLink = System.UInt64;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 64 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
19     public unsafe class UInt64SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20     {
21         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25         private LinksHeader<ulong>* _header;
26         private RawLinkDataPart<ulong>* _linksDataParts;
27         private RawLinkIndexPart<ulong>* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="dataMemory">
36         /// <para>A data memory.</para>
37         /// <para></para>

```

```

38     /// </param>
39     /// <param name="indexMemory">
40     /// <para>A index memory.</para>
41     /// <para></para>
42     /// </param>
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46     /// <summary>
47     /// <para>
48     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="dataMemory">
53     /// <para>A data memory.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="indexMemory">
57     /// <para>A index memory.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="memoryReservationStep">
61     /// <para>A memory reservation step.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }
66
67     /// <summary>
68     /// <para>
69     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="dataMemory">
74     /// <para>A data memory.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="indexMemory">
78     /// <para>A index memory.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="memoryReservationStep">
82     /// <para>A memory reservation step.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="constants">
86     /// <para>A constants.</para>
87     /// <para></para>
88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
        ↳ this(dataMemory, indexMemory, memoryReservationStep, constants,
        ↳ IndexTreeType.Default, useLinkedList: true) { }
91
92     /// <summary>
93     /// <para>
94     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="dataMemory">
99     /// <para>A data memory.</para>
100    /// <para></para>
101    /// </param>
102    /// <param name="indexMemory">
103    /// <para>A index memory.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="memoryReservationStep">
107    /// <para>A memory reservation step.</para>

```

```

108     /// <para></para>
109     /// </param>
110     /// <param name="constants">
111     /// <para>A constants.</para>
112     /// <para></para>
113     /// </param>
114     /// <param name="indexTreeType">
115     /// <para>A index tree type.</para>
116     /// <para></para>
117     /// </param>
118     /// <param name="useLinkedList">
119     /// <para>A use linked list.</para>
120     /// <para></para>
121     /// </param>
122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
123     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
        ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪ memoryReservationStep, constants, useLinkedList)
124     {
125         if (indexTreeType == IndexTreeType.SizeBalancedTree)
126         {
127             _createInternalSourceTreeMethods = () => new
        ↪ UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
128             _createExternalSourceTreeMethods = () => new
        ↪ UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
129             _createInternalTargetTreeMethods = () => new
        ↪ UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
130             _createExternalTargetTreeMethods = () => new
        ↪ UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
131         }
132         else
133         {
134             _createInternalSourceTreeMethods = () => new
        ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
135             _createExternalSourceTreeMethods = () => new
        ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
136             _createInternalTargetTreeMethods = () => new
        ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
137             _createExternalTargetTreeMethods = () => new
        ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
138         }
139         Init(dataMemory, indexMemory);
140     }
141
142     /// <summary>
143     /// <para>
144     /// Sets the pointers using the specified data memory.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="dataMemory">
149     /// <para>The data memory.</para>
150     /// <para></para>
151     /// </param>
152     /// <param name="indexMemory">
153     /// <para>The index memory.</para>
154     /// <para></para>
155     /// </param>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override void SetPointers(IResizableDirectMemory dataMemory,
        ↪ IResizableDirectMemory indexMemory)
158     {
159         _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
160         _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
161         _header = (LinksHeader<TLink>*)indexMemory.Pointer;
162         if (_useLinkedList)
163         {

```

```

164         InternalSourcesListMethods = new
165         ↪ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
166         ↪ _linksIndexParts);
167     }
168     else
169     {
170         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
171     }
172     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
173     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
174     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
175     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
176 }
177
178 /// <summary>
179 /// <para>
180 /// Resets the pointers.
181 /// </para>
182 /// <para></para>
183 /// </summary>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected override void ResetPointers()
186 {
187     base.ResetPointers();
188     _linksDataParts = null;
189     _linksIndexParts = null;
190     _header = null;
191 }
192
193 /// <summary>
194 /// <para>
195 /// Gets the header reference.
196 /// </para>
197 /// <para></para>
198 /// </summary>
199 /// <returns>
200 /// <para>A ref links header of t link</para>
201 /// <para></para>
202 /// </returns>
203 [MethodImpl(MethodImplOptions.AggressiveInlining)]
204 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
205
206 /// <summary>
207 /// <para>
208 /// Gets the link data part reference using the specified link index.
209 /// </para>
210 /// <para></para>
211 /// </summary>
212 /// <param name="linkIndex">
213 /// <para>The link index.</para>
214 /// <para></para>
215 /// </param>
216 /// <returns>
217 /// <para>A ref raw link data part of t link</para>
218 /// <para></para>
219 /// </returns>
220 [MethodImpl(MethodImplOptions.AggressiveInlining)]
221 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
222 ↪ => ref _linksDataParts[linkIndex];
223
224 /// <summary>
225 /// <para>
226 /// Gets the link index part reference using the specified link index.
227 /// </para>
228 /// <para></para>
229 /// </summary>
230 /// <param name="linkIndex">
231 /// <para>The link index.</para>
232 /// <para></para>
233 /// </param>
234 /// <returns>
235 /// <para>A ref raw link index part of t link</para>
236 /// <para></para>
237 /// </returns>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
240 ↪ linkIndex) => ref _linksIndexParts[linkIndex];

```

```

238     /// <summary>
239     /// <para>
240     /// Determines whether this instance are equal.
241     /// </para>
242     /// <para></para>
243     /// </summary>
244     /// <param name="first">
245     /// <para>The first.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="second">
249     /// <para>The second.</para>
250     /// <para></para>
251     /// </param>
252     /// <returns>
253     /// <para>The bool</para>
254     /// <para></para>
255     /// </returns>
256     [MethodImpl(MethodImplOptions.AggressiveInlining)]
257     protected override bool AreEqual(ulong first, ulong second) => first == second;
258
259     /// <summary>
260     /// <para>
261     /// Determines whether this instance less than.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="first">
266     /// <para>The first.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="second">
270     /// <para>The second.</para>
271     /// <para></para>
272     /// </param>
273     /// <returns>
274     /// <para>The bool</para>
275     /// <para></para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override bool LessThan(ulong first, ulong second) => first < second;
279
280     /// <summary>
281     /// <para>
282     /// Determines whether this instance less or equal than.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <param name="first">
287     /// <para>The first.</para>
288     /// <para></para>
289     /// </param>
290     /// <param name="second">
291     /// <para>The second.</para>
292     /// <para></para>
293     /// </param>
294     /// <returns>
295     /// <para>The bool</para>
296     /// <para></para>
297     /// </returns>
298     [MethodImpl(MethodImplOptions.AggressiveInlining)]
299     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
300
301     /// <summary>
302     /// <para>
303     /// Determines whether this instance greater than.
304     /// </para>
305     /// <para></para>
306     /// </summary>
307     /// <param name="first">
308     /// <para>The first.</para>
309     /// <para></para>
310     /// </param>
311     /// <param name="second">
312     /// <para>The second.</para>
313     /// <para></para>
314     /// </param>
315     /// </returns>

```



```

316     /// <para>The bool</para>
317     /// <para></para>
318     /// </returns>
319     [MethodImpl(MethodImplOptions.AggressiveInlining)]
320     protected override bool GreaterThan(ulong first, ulong second) => first > second;
321
322     /// <summary>
323     /// <para>
324     /// Determines whether this instance greater or equal than.
325     /// </para>
326     /// <para></para>
327     /// </summary>
328     /// <param name="first">
329     /// <para>The first.</para>
330     /// <para></para>
331     /// </param>
332     /// <param name="second">
333     /// <para>The second.</para>
334     /// <para></para>
335     /// </param>
336     /// <returns>
337     /// <para>The bool</para>
338     /// <para></para>
339     /// </returns>
340     [MethodImpl(MethodImplOptions.AggressiveInlining)]
341     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
342
343     /// <summary>
344     /// <para>
345     /// Gets the zero.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <returns>
350     /// <para>The ulong</para>
351     /// <para></para>
352     /// </returns>
353     [MethodImpl(MethodImplOptions.AggressiveInlining)]
354     protected override ulong GetZero() => 0UL;
355
356     /// <summary>
357     /// <para>
358     /// Gets the one.
359     /// </para>
360     /// <para></para>
361     /// </summary>
362     /// <returns>
363     /// <para>The ulong</para>
364     /// <para></para>
365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override ulong GetOne() => 1UL;
368
369     /// <summary>
370     /// <para>
371     /// Converts the to int 64 using the specified value.
372     /// </para>
373     /// <para></para>
374     /// </summary>
375     /// <param name="value">
376     /// <para>The value.</para>
377     /// <para></para>
378     /// </param>
379     /// <returns>
380     /// <para>The long</para>
381     /// <para></para>
382     /// </returns>
383     [MethodImpl(MethodImplOptions.AggressiveInlining)]
384     protected override long ConvertToInt64(ulong value) => (long)value;
385
386     /// <summary>
387     /// <para>
388     /// Converts the to address using the specified value.
389     /// </para>
390     /// <para></para>
391     /// </summary>
392     /// <param name="value">
393     /// <para>The value.</para>

```

```

394     /// <para></para>
395     /// </param>
396     /// <returns>
397     /// <para>The ulong</para>
398     /// <para></para>
399     /// </returns>
400     [MethodImpl(MethodImplOptions.AggressiveInlining)]
401     protected override ulong ConvertToAddress(long value) => (ulong)value;
402
403     /// <summary>
404     /// <para>
405     /// Adds the first.
406     /// </para>
407     /// <para></para>
408     /// </summary>
409     /// <param name="first">
410     /// <para>The first.</para>
411     /// <para></para>
412     /// </param>
413     /// <param name="second">
414     /// <para>The second.</para>
415     /// <para></para>
416     /// </param>
417     /// <returns>
418     /// <para>The ulong</para>
419     /// <para></para>
420     /// </returns>
421     [MethodImpl(MethodImplOptions.AggressiveInlining)]
422     protected override ulong Add(ulong first, ulong second) => first + second;
423
424     /// <summary>
425     /// <para>
426     /// Subtracts the first.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The ulong</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override ulong Subtract(ulong first, ulong second) => first - second;
444
445     /// <summary>
446     /// <para>
447     /// Increments the link.
448     /// </para>
449     /// <para></para>
450     /// </summary>
451     /// <param name="link">
452     /// <para>The link.</para>
453     /// <para></para>
454     /// </param>
455     /// <returns>
456     /// <para>The ulong</para>
457     /// <para></para>
458     /// </returns>
459     [MethodImpl(MethodImplOptions.AggressiveInlining)]
460     protected override ulong Increment(ulong link) => ++link;
461
462     /// <summary>
463     /// <para>
464     /// Decrements the link.
465     /// </para>
466     /// <para></para>
467     /// </summary>
468     /// <param name="link">
469     /// <para>The link.</para>
470     /// <para></para>
471     /// </param>

```

```

472     /// <returns>
473     /// <para>The ulong</para>
474     /// <para></para>
475     /// </returns>
476     [MethodImpl(MethodImplOptions.AggressiveInlining)]
477     protected override ulong Decrement(ulong link) => --link;
478 }
479 }

```

1.79 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLink}" />
16     public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<TLink>
17     {
18         private readonly RawLinkDataPart<ulong>* _links;
19         private readonly LinksHeader<ulong>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt64UnusedLinksListMethods" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
37         ↪ header)
38             : base((byte*)links, (byte*)header)
39         {
40             _links = links;
41             _header = header;
42         }
43
44         /// <summary>
45         /// <para>
46         /// Gets the link data part reference using the specified link.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="link">
51         /// <para>The link.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>A ref raw link data part of t link</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
60         ↪ ref _links[link];
61
62         /// <summary>
63         /// <para>
64         /// Gets the header reference.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         /// <returns>
69         /// <para>A ref links header of t link</para>

```

```

68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
72 }
73 }

```

1.80 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using Platform.Numbers;
9  using static System.Runtime.CompilerServices.Unsafe;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Memory.United.Generic
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the links avl balanced tree methods base.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     /// <seealso cref="SizedAndThreadedAVLBalancedTreeMethods{TLink}"/>
22     /// <seealso cref="ILinksTreeMethods{TLink}"/>
23     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLink> :
24         ↳ SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
25     {
26         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
27             ↳ UncheckedConverter<TLink, long>.Default;
28         private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
29             ↳ UncheckedConverter<TLink, int>.Default;
30         private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
31             ↳ UncheckedConverter<bool, TLink>.Default;
32         private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
33             ↳ UncheckedConverter<TLink, bool>.Default;
34         private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
35             ↳ UncheckedConverter<int, TLink>.Default;
36
37         /// <summary>
38         /// <para>
39         /// The break.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected readonly TLink Break;
44
45         /// <summary>
46         /// <para>
47         /// The continue.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         protected readonly TLink Continue;
52
53         /// <summary>
54         /// <para>
55         /// The links.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         protected readonly byte* Links;
60
61         /// <summary>
62         /// <para>
63         /// The header.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         protected readonly byte* Header;
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         /// <param name="constants">

```

```

67     /// <para>A constants.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="links">
71     /// <para>A links.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="header">
75     /// <para>A header.</para>
76     /// <para></para>
77     /// </param>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
80     ↪ byte* header)
81     {
82         Links = links;
83         Header = header;
84         Break = constants.Break;
85         Continue = constants.Continue;
86     }
87     /// <summary>
88     /// <para>
89     /// Gets the tree root.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <returns>
94     /// <para>The link</para>
95     /// <para></para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected abstract TLink GetTreeRoot();
99
100    /// <summary>
101    /// <para>
102    /// Gets the base part value using the specified link.
103    /// </para>
104    /// <para></para>
105    /// </summary>
106    /// <param name="link">
107    /// <para>The link.</para>
108    /// <para></para>
109    /// </param>
110    /// <returns>
111    /// <para>The link</para>
112    /// <para></para>
113    /// </returns>
114    [MethodImpl(MethodImplOptions.AggressiveInlining)]
115    protected abstract TLink GetBasePartValue(TLink link);
116
117    /// <summary>
118    /// <para>
119    /// Determines whether this instance first is to the right of second.
120    /// </para>
121    /// <para></para>
122    /// </summary>
123    /// <param name="source">
124    /// <para>The source.</para>
125    /// <para></para>
126    /// </param>
127    /// <param name="target">
128    /// <para>The target.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="rootSource">
132    /// <para>The root source.</para>
133    /// <para></para>
134    /// </param>
135    /// <param name="rootTarget">
136    /// <para>The root target.</para>
137    /// <para></para>
138    /// </param>
139    /// <returns>
140    /// <para>The bool</para>
141    /// <para></para>
142    /// </returns>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

144     protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪     rootSource, TLink rootTarget);
145
146     /// <summary>
147     /// <para>
148     /// Determines whether this instance first is to the left of second.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="source">
153     /// <para>The source.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="target">
157     /// <para>The target.</para>
158     /// <para></para>
159     /// </param>
160     /// <param name="rootSource">
161     /// <para>The root source.</para>
162     /// <para></para>
163     /// </param>
164     /// <param name="rootTarget">
165     /// <para>The root target.</para>
166     /// <para></para>
167     /// </param>
168     /// <returns>
169     /// <para>The bool</para>
170     /// <para></para>
171     /// </returns>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪     rootSource, TLink rootTarget);
174
175     /// <summary>
176     /// <para>
177     /// Gets the header reference.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <returns>
182     /// <para>A ref links header of t link</para>
183     /// <para></para>
184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪     AsRef<LinksHeader<TLink>>(Header);
187
188     /// <summary>
189     /// <para>
190     /// Gets the link reference using the specified link.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     /// <param name="link">
195     /// <para>The link.</para>
196     /// <para></para>
197     /// </param>
198     /// <returns>
199     /// <para>A ref raw link of t link</para>
200     /// <para></para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
    ↪     AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
    ↪     _addressToInt64Converter.Convert(link)));
204
205     /// <summary>
206     /// <para>
207     /// Gets the link values using the specified link index.
208     /// </para>
209     /// <para></para>
210     /// </summary>
211     /// <param name="linkIndex">
212     /// <para>The link index.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>

```

```

216 /// <para>A list of t link</para>
217 /// <para></para>
218 /// </returns>
219 [MethodImpl(MethodImplOptions.AggressiveInlining)]
220 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
221 {
222     ref var link = ref GetLinkReference(linkIndex);
223     return new Link<TLink>(linkIndex, link.Source, link.Target);
224 }
225
226 /// <summary>
227 /// <para>
228 /// Determines whether this instance first is to the left of second.
229 /// </para>
230 /// <para></para>
231 /// </summary>
232 /// <param name="first">
233 /// <para>The first.</para>
234 /// <para></para>
235 /// </param>
236 /// <param name="second">
237 /// <para>The second.</para>
238 /// <para></para>
239 /// </param>
240 /// <returns>
241 /// <para>The bool</para>
242 /// <para></para>
243 /// </returns>
244 [MethodImpl(MethodImplOptions.AggressiveInlining)]
245 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
246 {
247     ref var firstLink = ref GetLinkReference(first);
248     ref var secondLink = ref GetLinkReference(second);
249     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
250         ↪ secondLink.Source, secondLink.Target);
251 }
252
253 /// <summary>
254 /// <para>
255 /// Determines whether this instance first is to the right of second.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="first">
260 /// <para>The first.</para>
261 /// <para></para>
262 /// </param>
263 /// <param name="second">
264 /// <para>The second.</para>
265 /// <para></para>
266 /// </param>
267 /// <returns>
268 /// <para>The bool</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
273 {
274     ref var firstLink = ref GetLinkReference(first);
275     ref var secondLink = ref GetLinkReference(second);
276     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
277         ↪ secondLink.Source, secondLink.Target);
278 }
279
280 /// <summary>
281 /// <para>
282 /// Gets the size value using the specified value.
283 /// </para>
284 /// <para></para>
285 /// </summary>
286 /// <param name="value">
287 /// <para>The value.</para>
288 /// <para></para>
289 /// </param>
290 /// <returns>
291 /// <para>The link</para>
292 /// <para></para>
293 /// </returns>

```

```

292 [MethodImpl(MethodImplOptions.AggressiveInlining)]
293 protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
    ↪ -5);
294
295 /// <summary>
296 /// <para>
297 /// Sets the size value using the specified stored value.
298 /// </para>
299 /// <para></para>
300 /// </summary>
301 /// <param name="storedValue">
302 /// <para>The stored value.</para>
303 /// <para></para>
304 /// </param>
305 /// <param name="size">
306 /// <para>The size.</para>
307 /// <para></para>
308 /// </param>
309 [MethodImpl(MethodImplOptions.AggressiveInlining)]
310 protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
    ↪ Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
311
312 /// <summary>
313 /// <para>
314 /// Determines whether this instance get left is child value.
315 /// </para>
316 /// <para></para>
317 /// </summary>
318 /// <param name="value">
319 /// <para>The value.</para>
320 /// <para></para>
321 /// </param>
322 /// <returns>
323 /// <para>The bool</para>
324 /// <para></para>
325 /// </returns>
326 [MethodImpl(MethodImplOptions.AggressiveInlining)]
327 protected virtual bool GetLeftIsChildValue(TLink value)
328 {
329     unchecked
330     {
331         return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
332         //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
333     }
334 }
335
336 /// <summary>
337 /// <para>
338 /// Sets the left is child value using the specified stored value.
339 /// </para>
340 /// <para></para>
341 /// </summary>
342 /// <param name="storedValue">
343 /// <para>The stored value.</para>
344 /// <para></para>
345 /// </param>
346 /// <param name="value">
347 /// <para>The value.</para>
348 /// <para></para>
349 /// </param>
350 [MethodImpl(MethodImplOptions.AggressiveInlining)]
351 protected virtual void SetLeftIsChildValue(ref TLink storedValue, bool value)
352 {
353     unchecked
354     {
355         var previousValue = storedValue;
356         var modified = Bit<TLink>.PartialWrite(previousValue,
    ↪ _boolToAddressConverter.Convert(value), 4, 1);
357         storedValue = modified;
358     }
359 }
360
361 /// <summary>
362 /// <para>
363 /// Determines whether this instance get right is child value.
364 /// </para>
365 /// <para></para>
366 /// </summary>

```



```

367     /// <param name="value">
368     /// <para>The value.</para>
369     /// <para></para>
370     /// </param>
371     /// <returns>
372     /// <para>The bool</para>
373     /// <para></para>
374     /// </returns>
375     [MethodImpl(MethodImplOptions.AggressiveInlining)]
376     protected virtual bool GetRightIsChildValue(TLink value)
377     {
378         unchecked
379         {
380             return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 3, 1));
381             //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
382         }
383     }
384
385     /// <summary>
386     /// <para>
387     /// Sets the right is child value using the specified stored value.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     /// <param name="storedValue">
392     /// <para>The stored value.</para>
393     /// <para></para>
394     /// </param>
395     /// <param name="value">
396     /// <para>The value.</para>
397     /// <para></para>
398     /// </param>
399     [MethodImpl(MethodImplOptions.AggressiveInlining)]
400     protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
401     {
402         unchecked
403         {
404             var previousValue = storedValue;
405             var modified = Bit<TLink>.PartialWrite(previousValue,
406                 ↪ _boolToAddressConverter.Convert(value), 3, 1);
407             storedValue = modified;
408         }
409     }
410
411     /// <summary>
412     /// <para>
413     /// Determines whether this instance is child.
414     /// </para>
415     /// <para></para>
416     /// </summary>
417     /// <param name="parent">
418     /// <para>The parent.</para>
419     /// <para></para>
420     /// </param>
421     /// <param name="possibleChild">
422     /// <para>The possible child.</para>
423     /// <para></para>
424     /// </param>
425     /// <returns>
426     /// <para>The bool</para>
427     /// <para></para>
428     /// </returns>
429     [MethodImpl(MethodImplOptions.AggressiveInlining)]
430     protected bool IsChild(TLink parent, TLink possibleChild)
431     {
432         var parentSize = GetSize(parent);
433         var childSize = GetSizeOrZero(possibleChild);
434         return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
435     }
436
437     /// <summary>
438     /// <para>
439     /// Gets the balance value using the specified stored value.
440     /// </para>
441     /// <para></para>
442     /// </summary>
443     /// <param name="storedValue">
444     /// <para>The stored value.</para>

```

```

444    /// <para></para>
445    /// </param>
446    /// <returns>
447    /// <para>The sbyte</para>
448    /// <para></para>
449    /// </returns>
450    [MethodImpl(MethodImplOptions.AggressiveInlining)]
451    protected virtual sbyte GetBalanceValue(TLink storedValue)
452    {
453        unchecked
454        {
455            var value = _addressToInt32Converter.Convert(Bit<TLink>.PartialRead(storedValue,
456                ↪ 0, 3));
457            value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
458                ↪ end of sbyte
459            return (sbyte)value;
460        }
461    }
462    /// <summary>
463    /// <para>
464    /// Sets the balance value using the specified stored value.
465    /// </para>
466    /// <para></para>
467    /// </summary>
468    /// <param name="storedValue">
469    /// <para>The stored value.</para>
470    /// <para></para>
471    /// </param>
472    /// <param name="value">
473    /// <para>The value.</para>
474    /// <para></para>
475    /// </param>
476    [MethodImpl(MethodImplOptions.AggressiveInlining)]
477    protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
478    {
479        unchecked
480        {
481            var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
482                ↪ value & 3);
483            var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
484            storedValue = modified;
485        }
486    }
487    /// <summary>
488    /// <para>
489    /// The zero.
490    /// </para>
491    /// <para></para>
492    /// </summary>
493    public TLink this[TLink index]
494    {
495        [MethodImpl(MethodImplOptions.AggressiveInlining)]
496        get
497        {
498            var root = GetTreeRoot();
499            if (GreaterOrEqualThan(index, GetSize(root)))
500            {
501                return Zero;
502            }
503            while (!EqualToZero(root))
504            {
505                var left = GetLeftOrDefault(root);
506                var leftSize = GetSizeOrZero(left);
507                if (LessThan(index, leftSize))
508                {
509                    root = left;
510                    continue;
511                }
512                if (AreEqual(index, leftSize))
513                {
514                    return root;
515                }
516                root = GetRightOrDefault(root);
517                index = Subtract(index, Increment(leftSize));
518            }
519        }
520    }

```

```

518         return Zero; // TODO: Impossible situation exception (only if tree structure
519         ↪ broken)
520     }
521 }
522
523 /// <summary>
524 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
525 ↪ (концом).
526 /// </summary>
527 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
528 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
529 /// <returns>Индекс искомой связи.</returns>
530 [MethodImpl(MethodImplOptions.AggressiveInlining)]
531 public TLink Search(TLink source, TLink target)
532 {
533     var root = GetTreeRoot();
534     while (!EqualToZero(root))
535     {
536         ref var rootLink = ref GetLinkReference(root);
537         var rootSource = rootLink.Source;
538         var rootTarget = rootLink.Target;
539         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
540             ↪ node.Key < root.Key
541         {
542             root = GetLeftOrDefault(root);
543         }
544         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
545             ↪ node.Key > root.Key
546         {
547             root = GetRightOrDefault(root);
548         }
549         else // node.Key == root.Key
550         {
551             return root;
552         }
553     }
554     return Zero;
555 }
556
557 // TODO: Return indices range instead of references count
558 /// <summary>
559 /// <para>
560 /// Counts the usages using the specified link.
561 /// </para>
562 /// <para></para>
563 /// </summary>
564 /// <param name="link">
565 /// <para>The link.</para>
566 /// <para></para>
567 /// </param>
568 /// <returns>
569 /// <para>The link</para>
570 /// <para></para>
571 /// </returns>
572 [MethodImpl(MethodImplOptions.AggressiveInlining)]
573 public TLink CountUsages(TLink link)
574 {
575     var root = GetTreeRoot();
576     var total = GetSize(root);
577     var totalRightIgnore = Zero;
578     while (!EqualToZero(root))
579     {
580         var @base = GetBasePartValue(root);
581         if (LessOrEqualThan(@base, link))
582         {
583             root = GetRightOrDefault(root);
584         }
585         else
586         {
587             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
588             root = GetLeftOrDefault(root);
589         }
590     }
591     root = GetTreeRoot();
592     var totalLeftIgnore = Zero;
593     while (!EqualToZero(root))
594     {
595         var @base = GetBasePartValue(root);

```

```

592         if (GreaterOrEqualThan(@base, link))
593         {
594             root = GetLeftOrDefault(root);
595         }
596         else
597         {
598             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
599
600             root = GetRightOrDefault(root);
601         }
602     }
603     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
604 }
605
606 /// <summary>
607 /// <para>
608 /// Eaches the usage using the specified link.
609 /// </para>
610 /// <para></para>
611 /// </summary>
612 /// <param name="link">
613 /// <para>The link.</para>
614 /// <para></para>
615 /// </param>
616 /// <param name="handler">
617 /// <para>The handler.</para>
618 /// <para></para>
619 /// </param>
620 /// <returns>
621 /// <para>The continue.</para>
622 /// <para></para>
623 /// </returns>
624 [MethodImpl(MethodImplOptions.AggressiveInlining)]
625 public TLink EachUsage(TLink link, ReadHandler<TLink>? handler)
626 {
627     var root = GetTreeRoot();
628     if (EqualToZero(root))
629     {
630         return Continue;
631     }
632     TLink first = Zero, current = root;
633     while (!EqualToZero(current))
634     {
635         var @base = GetBasePartValue(current);
636         if (GreaterOrEqualThan(@base, link))
637         {
638             if (AreEqual(@base, link))
639             {
640                 first = current;
641             }
642             current = GetLeftOrDefault(current);
643         }
644         else
645         {
646             current = GetRightOrDefault(current);
647         }
648     }
649     if (!EqualToZero(first))
650     {
651         current = first;
652         while (true)
653         {
654             if (AreEqual(handler(GetLinkValues(current)), Break))
655             {
656                 return Break;
657             }
658             current = GetNext(current);
659             if (EqualToZero(current) || !AreEqual(GetBasePartValue(current), link))
660             {
661                 break;
662             }
663         }
664     }
665     return Continue;
666 }
667
668 /// <summary>
669 /// <para>
670 /// Prints the node value using the specified node.

```

```

671     /// </para>
672     /// <para></para>
673     /// </summary>
674     /// <param name="node">
675     /// <para>The node.</para>
676     /// <para></para>
677     /// </param>
678     /// <param name="sb">
679     /// <para>The sb.</para>
680     /// <para></para>
681     /// </param>
682     [MethodImpl(MethodImplOptions.AggressiveInlining)]
683     protected override void PrintNodeValue(TLink node, StringBuilder sb)
684     {
685         ref var link = ref GetLinkReference(node);
686         sb.Append(' ');
687         sb.Append(link.Source);
688         sb.Append('-');
689         sb.Append('>');
690         sb.Append(link.Target);
691     }
692 }
693 }

```

1.81 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.United.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}" />
21     /// <seealso cref="ILinksTreeMethods{TLink}" />
22     public unsafe abstract class LinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
23     ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26         ↪ UncheckedConverter<TLink, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// The links.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* Links;
51
52         /// <summary>
53         /// <para>
54         /// The header.
55         /// </para>
56         /// <para></para>
57         /// </summary>

```

```

53     protected readonly byte* Header;
54
55     /// <summary>
56     /// <para>
57     /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
58     ↪ instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="constants">
63     /// <para>A constants.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="links">
67     /// <para>A links.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="header">
71     /// <para>A header.</para>
72     /// <para></para>
73     /// </param>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
76     ↪ byte* links, byte* header)
77     {
78         Links = links;
79         Header = header;
80         Break = constants.Break;
81         Continue = constants.Continue;
82     }
83
84     /// <summary>
85     /// <para>
86     /// Gets the tree root.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <returns>
91     /// <para>The link</para>
92     /// <para></para>
93     /// </returns>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     protected abstract TLink GetTreeRoot();
96
97     /// <summary>
98     /// <para>
99     /// Gets the base part value using the specified link.
100    /// </para>
101    /// <para></para>
102    /// </summary>
103    /// <param name="link">
104    /// <para>The link.</para>
105    /// <para></para>
106    /// </param>
107    /// <returns>
108    /// <para>The link</para>
109    /// <para></para>
110    /// </returns>
111    [MethodImpl(MethodImplOptions.AggressiveInlining)]
112    protected abstract TLink GetBasePartValue(TLink link);
113
114    /// <summary>
115    /// <para>
116    /// Determines whether this instance first is to the right of second.
117    /// </para>
118    /// <para></para>
119    /// </summary>
120    /// <param name="source">
121    /// <para>The source.</para>
122    /// <para></para>
123    /// </param>
124    /// <param name="target">
125    /// <para>The target.</para>
126    /// <para></para>
127    /// </param>
128    /// <param name="rootSource">
129    /// <para>The root source.</para>
130    /// <para></para>
131    /// </param>

```

```

129    /// </param>
130    /// <param name="rootTarget">
131    /// <para>The root target.</para>
132    /// <para></para>
133    /// </param>
134    /// <returns>
135    /// <para>The bool</para>
136    /// <para></para>
137    /// </returns>
138    [MethodImpl(MethodImplOptions.AggressiveInlining)]
139    protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);

140
141    /// <summary>
142    /// <para>
143    /// Determines whether this instance first is to the left of second.
144    /// </para>
145    /// <para></para>
146    /// </summary>
147    /// <param name="source">
148    /// <para>The source.</para>
149    /// <para></para>
150    /// </param>
151    /// <param name="target">
152    /// <para>The target.</para>
153    /// <para></para>
154    /// </param>
155    /// <param name="rootSource">
156    /// <para>The root source.</para>
157    /// <para></para>
158    /// </param>
159    /// <param name="rootTarget">
160    /// <para>The root target.</para>
161    /// <para></para>
162    /// </param>
163    /// <returns>
164    /// <para>The bool</para>
165    /// <para></para>
166    /// </returns>
167    [MethodImpl(MethodImplOptions.AggressiveInlining)]
168    protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);

169
170    /// <summary>
171    /// <para>
172    /// Gets the header reference.
173    /// </para>
174    /// <para></para>
175    /// </summary>
176    /// <returns>
177    /// <para>A ref links header of t link</para>
178    /// <para></para>
179    /// </returns>
180    [MethodImpl(MethodImplOptions.AggressiveInlining)]
181    protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪ AsRef<LinksHeader<TLink>>(Header);

182
183    /// <summary>
184    /// <para>
185    /// Gets the link reference using the specified link.
186    /// </para>
187    /// <para></para>
188    /// </summary>
189    /// <param name="link">
190    /// <para>The link.</para>
191    /// <para></para>
192    /// </param>
193    /// <returns>
194    /// <para>A ref raw link of t link</para>
195    /// <para></para>
196    /// </returns>
197    [MethodImpl(MethodImplOptions.AggressiveInlining)]
198    protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));

199
200    /// <summary>

```

```

201 /// <para>
202 /// Gets the link values using the specified link index.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="linkIndex">
207 /// <para>The link index.</para>
208 /// <para></para>
209 /// </param>
210 /// <returns>
211 /// <para>A list of t link</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
216 {
217     ref var link = ref GetLinkReference(linkIndex);
218     return new Link<TLink>(linkIndex, link.Source, link.Target);
219 }
220
221 /// <summary>
222 /// <para>
223 /// Determines whether this instance first is to the left of second.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="first">
228 /// <para>The first.</para>
229 /// <para></para>
230 /// </param>
231 /// <param name="second">
232 /// <para>The second.</para>
233 /// <para></para>
234 /// </param>
235 /// <returns>
236 /// <para>The bool</para>
237 /// <para></para>
238 /// </returns>
239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
245         ↪ secondLink.Source, secondLink.Target);
246 }
247
248 /// <summary>
249 /// <para>
250 /// Determines whether this instance first is to the right of second.
251 /// </para>
252 /// <para></para>
253 /// </summary>
254 /// <param name="first">
255 /// <para>The first.</para>
256 /// <para></para>
257 /// </param>
258 /// <param name="second">
259 /// <para>The second.</para>
260 /// <para></para>
261 /// </param>
262 /// <returns>
263 /// <para>The bool</para>
264 /// <para></para>
265 /// </returns>
266 [MethodImpl(MethodImplOptions.AggressiveInlining)]
267 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
268 {
269     ref var firstLink = ref GetLinkReference(first);
270     ref var secondLink = ref GetLinkReference(second);
271     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
272         ↪ secondLink.Source, secondLink.Target);
273 }
274
275 /// <summary>
276 /// <para>
277 /// The zero.
278 /// </para>

```



```

277 /// <para></para>
278 /// </summary>
279 public TLink this[TLink index]
280 {
281     [MethodImpl(MethodImplOptions.AggressiveInlining)]
282     get
283     {
284         var root = GetTreeRoot();
285         if (GreaterOrEqualThan(index, GetSize(root)))
286         {
287             return Zero;
288         }
289         while (!EqualToZero(root))
290         {
291             var left = GetLeftOrDefault(root);
292             var leftSize = GetSizeOrZero(left);
293             if (LessThan(index, leftSize))
294             {
295                 root = left;
296                 continue;
297             }
298             if (AreEqual(index, leftSize))
299             {
300                 return root;
301             }
302             root = GetRightOrDefault(root);
303             index = Subtract(index, Increment(leftSize));
304         }
305         return Zero; // TODO: Impossible situation exception (only if tree structure
306                       ↪ broken)
307     }
308 }
309
310 /// <summary>
311 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
312 /// ↪ (концом).
313 /// </summary>
314 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
315 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
316 /// <returns>Индекс искомой связи.</returns>
317 [MethodImpl(MethodImplOptions.AggressiveInlining)]
318 public TLink Search(TLink source, TLink target)
319 {
320     var root = GetTreeRoot();
321     while (!EqualToZero(root))
322     {
323         ref var rootLink = ref GetLinkReference(root);
324         var rootSource = rootLink.Source;
325         var rootTarget = rootLink.Target;
326         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
327             ↪ node.Key < root.Key
328         {
329             root = GetLeftOrDefault(root);
330         }
331         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
332             ↪ node.Key > root.Key
333         {
334             root = GetRightOrDefault(root);
335         }
336         else // node.Key == root.Key
337         {
338             return root;
339         }
340     }
341     return Zero;
342 }
343
344 // TODO: Return indices range instead of references count
345 /// <summary>
346 /// <para>
347 /// Counts the usages using the specified link.
348 /// </para>
349 /// <para></para>
350 /// </summary>
351 /// <param name="link">
352 /// <para>The link.</para>
353 /// </param>

```

```

351     /// <returns>
352     /// <para>The link</para>
353     /// <para></para>
354     /// </returns>
355     [MethodImpl(MethodImplOptions.AggressiveInlining)]
356     public TLink CountUsages(TLink link)
357     {
358         var root = GetTreeRoot();
359         var total = GetSize(root);
360         var totalRightIgnore = Zero;
361         while (!EqualToZero(root))
362         {
363             var @base = GetBasePartValue(root);
364             if (LessOrEqualThan(@base, link))
365             {
366                 root = GetRightOrDefault(root);
367             }
368             else
369             {
370                 totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
371                 root = GetLeftOrDefault(root);
372             }
373         }
374         root = GetTreeRoot();
375         var totalLeftIgnore = Zero;
376         while (!EqualToZero(root))
377         {
378             var @base = GetBasePartValue(root);
379             if (GreaterOrEqualThan(@base, link))
380             {
381                 root = GetLeftOrDefault(root);
382             }
383             else
384             {
385                 totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
386                 root = GetRightOrDefault(root);
387             }
388         }
389         return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
390     }
391
392     /// <summary>
393     /// <para>
394     /// Eaches the usage using the specified base.
395     /// </para>
396     /// <para></para>
397     /// </summary>
398     /// <param name="@base">
399     /// <para>The base.</para>
400     /// <para></para>
401     /// </param>
402     /// <param name="handler">
403     /// <para>The handler.</para>
404     /// <para></para>
405     /// </param>
406     /// <returns>
407     /// <para>The link</para>
408     /// <para></para>
409     /// </returns>
410     [MethodImpl(MethodImplOptions.AggressiveInlining)]
411     public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
412         ↪ GetTreeRoot(), handler);
413
414     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
415     ↪ low-level MSIL stack.
416     [MethodImpl(MethodImplOptions.AggressiveInlining)]
417     private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
418     {
419         var @continue = Continue;
420         if (EqualToZero(link))
421         {
422             return @continue;
423         }
424         var linkBasePart = GetBasePartValue(link);
425         var @break = Break;
426         if (GreaterThan(linkBasePart, @base))
427         {
428             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))

```

```

427         {
428             return @break;
429         }
430     }
431     else if (LessThan(linkBasePart, @base))
432     {
433         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
434         {
435             return @break;
436         }
437     }
438     else //if (linkBasePart == @base)
439     {
440         if (AreEqual(handler(GetLinkValues(link)), @break))
441         {
442             return @break;
443         }
444         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
445         {
446             return @break;
447         }
448         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
449         {
450             return @break;
451         }
452     }
453     return @continue;
454 }
455
456 /// <summary>
457 /// <para>
458 /// Prints the node value using the specified node.
459 /// </para>
460 /// <para></para>
461 /// </summary>
462 /// <param name="node">
463 /// <para>The node.</para>
464 /// <para></para>
465 /// </param>
466 /// <param name="sb">
467 /// <para>The sb.</para>
468 /// <para></para>
469 /// </param>
470 [MethodImpl(MethodImplOptions.AggressiveInlining)]
471 protected override void PrintNodeValue(TLink node, StringBuilder sb)
472 {
473     ref var link = ref GetLinkReference(node);
474     sb.Append(' ');
475     sb.Append(link.Source);
476     sb.Append('-');
477     sb.Append('>');
478     sb.Append(link.Target);
479 }
480 }
481 }

```

1.82 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.United.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
21     /// <seealso cref="ILinksTreeMethods{TLink}"/>
22     public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLink> :
        ↳ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>

```

```

23 {
24     private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
25         ↳ UncheckedConverter<TLink, long>.Default;
26
27     /// <summary>
28     /// <para>
29     /// The break.
30     /// </para>
31     /// </summary>
32     protected readonly TLink Break;
33     /// <summary>
34     /// <para>
35     /// The continue.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected readonly TLink Continue;
40     /// <summary>
41     /// <para>
42     /// The links.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     protected readonly byte* Links;
47     /// <summary>
48     /// <para>
49     /// The header.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     protected readonly byte* Header;
54
55     /// <summary>
56     /// <para>
57     /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase"/> instance.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="constants">
62     /// <para>A constants.</para>
63     /// <para></para>
64     /// </param>
65     /// <param name="links">
66     /// <para>A links.</para>
67     /// <para></para>
68     /// </param>
69     /// <param name="header">
70     /// <para>A header.</para>
71     /// <para></para>
72     /// </param>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected LinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
75         ↳ byte* header)
76     {
77         Links = links;
78         Header = header;
79         Break = constants.Break;
80         Continue = constants.Continue;
81     }
82
83     /// <summary>
84     /// <para>
85     /// Gets the tree root.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <returns>
90     /// <para>The link</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     protected abstract TLink GetTreeRoot();
95
96     /// <summary>
97     /// <para>
98     /// Gets the base part value using the specified link.
99     /// </para>
100    /// <para></para>

```

```

100     /// </summary>
101     /// <param name="link">
102     /// <para>The link.</para>
103     /// <para></para>
104     /// </param>
105     /// <returns>
106     /// <para>The link</para>
107     /// <para></para>
108     /// </returns>
109     [MethodImpl(MethodImplOptions.AggressiveInlining)]
110     protected abstract TLink GetBasePartValue(TLink link);
111
112     /// <summary>
113     /// <para>
114     /// <para>Determines whether this instance first is to the right of second.
115     /// </para>
116     /// <para></para>
117     /// </summary>
118     /// <param name="source">
119     /// <para>The source.</para>
120     /// <para></para>
121     /// </param>
122     /// <param name="target">
123     /// <para>The target.</para>
124     /// <para></para>
125     /// </param>
126     /// <param name="rootSource">
127     /// <para>The root source.</para>
128     /// <para></para>
129     /// </param>
130     /// <param name="rootTarget">
131     /// <para>The root target.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The bool</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);
140
141     /// <summary>
142     /// <para>
143     /// <para>Determines whether this instance first is to the left of second.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="source">
148     /// <para>The source.</para>
149     /// <para></para>
150     /// </param>
151     /// <param name="target">
152     /// <para>The target.</para>
153     /// <para></para>
154     /// </param>
155     /// <param name="rootSource">
156     /// <para>The root source.</para>
157     /// <para></para>
158     /// </param>
159     /// <param name="rootTarget">
160     /// <para>The root target.</para>
161     /// <para></para>
162     /// </param>
163     /// <returns>
164     /// <para>The bool</para>
165     /// <para></para>
166     /// </returns>
167     [MethodImpl(MethodImplOptions.AggressiveInlining)]
168     protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);
169
170     /// <summary>
171     /// <para>
172     /// <para>Gets the header reference.
173     /// </para>
174     /// <para></para>
175     /// </summary>

```

```

176    /// <returns>
177    /// <para>A ref links header of t link</para>
178    /// <para></para>
179    /// </returns>
180    [MethodImpl(MethodImplOptions.AggressiveInlining)]
181    protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪ AsRef<LinksHeader<TLink>>(Header);

182
183    /// <summary>
184    /// <para>
185    /// Gets the link reference using the specified link.
186    /// </para>
187    /// <para></para>
188    /// </summary>
189    /// <param name="link">
190    /// <para>The link.</para>
191    /// <para></para>
192    /// </param>
193    /// <returns>
194    /// <para>A ref raw link of t link</para>
195    /// <para></para>
196    /// </returns>
197    [MethodImpl(MethodImplOptions.AggressiveInlining)]
198    protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));

199
200    /// <summary>
201    /// <para>
202    /// Gets the link values using the specified link index.
203    /// </para>
204    /// <para></para>
205    /// </summary>
206    /// <param name="linkIndex">
207    /// <para>The link index.</para>
208    /// <para></para>
209    /// </param>
210    /// <returns>
211    /// <para>A list of t link</para>
212    /// <para></para>
213    /// </returns>
214    [MethodImpl(MethodImplOptions.AggressiveInlining)]
215    protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
216    {
217        ref var link = ref GetLinkReference(linkIndex);
218        return new Link<TLink>(linkIndex, link.Source, link.Target);
219    }

220
221    /// <summary>
222    /// <para>
223    /// Determines whether this instance first is to the left of second.
224    /// </para>
225    /// <para></para>
226    /// </summary>
227    /// <param name="first">
228    /// <para>The first.</para>
229    /// <para></para>
230    /// </param>
231    /// <param name="second">
232    /// <para>The second.</para>
233    /// <para></para>
234    /// </param>
235    /// <returns>
236    /// <para>The bool</para>
237    /// <para></para>
238    /// </returns>
239    [MethodImpl(MethodImplOptions.AggressiveInlining)]
240    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
241    {
242        ref var firstLink = ref GetLinkReference(first);
243        ref var secondLink = ref GetLinkReference(second);
244        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
245    }

246
247    /// <summary>
248    /// <para>
249    /// Determines whether this instance first is to the right of second.

```

```

250    /// </para>
251    /// <para></para>
252    /// </summary>
253    /// <param name="first">
254    /// <para>The first.</para>
255    /// <para></para>
256    /// </param>
257    /// <param name="second">
258    /// <para>The second.</para>
259    /// <para></para>
260    /// </param>
261    /// <returns>
262    /// <para>The bool</para>
263    /// <para></para>
264    /// </returns>
265    [MethodImpl(MethodImplOptions.AggressiveInlining)]
266    protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
267    {
268        ref var firstLink = ref GetLinkReference(first);
269        ref var secondLink = ref GetLinkReference(second);
270        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
271            ↪ secondLink.Source, secondLink.Target);
272    }
273    /// <summary>
274    /// <para>
275    /// The zero.
276    /// </para>
277    /// <para></para>
278    /// </summary>
279    public TLink this[TLink index]
280    {
281        [MethodImpl(MethodImplOptions.AggressiveInlining)]
282        get
283        {
284            var root = GetTreeRoot();
285            if (GreaterOrEqualThan(index, GetSize(root)))
286            {
287                return Zero;
288            }
289            while (!EqualToZero(root))
290            {
291                var left = GetLeftOrDefault(root);
292                var leftSize = GetSizeOrZero(left);
293                if (LessThan(index, leftSize))
294                {
295                    root = left;
296                    continue;
297                }
298                if (AreEqual(index, leftSize))
299                {
300                    return root;
301                }
302                root = GetRightOrDefault(root);
303                index = Subtract(index, Increment(leftSize));
304            }
305            return Zero; // TODO: Impossible situation exception (only if tree structure
306                ↪ broken)
307        }
308    }
309    /// <summary>
310    /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
311    ↪ (концом).
312    /// </summary>
313    /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
314    /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
315    /// <returns>Индекс искомой связи.</returns>
316    [MethodImpl(MethodImplOptions.AggressiveInlining)]
317    public TLink Search(TLink source, TLink target)
318    {
319        var root = GetTreeRoot();
320        while (!EqualToZero(root))
321        {
322            ref var rootLink = ref GetLinkReference(root);
323            var rootSource = rootLink.Source;
324            var rootTarget = rootLink.Target;

```

```

324         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
325             ↪ node.Key < root.Key
326         {
327             root = GetLeftOrDefault(root);
328         }
329         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
330             ↪ node.Key > root.Key
331         {
332             root = GetRightOrDefault(root);
333         }
334         else // node.Key == root.Key
335         {
336             return root;
337         }
338     }
339     return Zero;
340 }
341
342 // TODO: Return indices range instead of references count
343 /// <summary>
344 /// <para>
345 /// Counts the usages using the specified link.
346 /// </para>
347 /// <para></para>
348 /// </summary>
349 /// <param name="link">
350 /// <para>The link.</para>
351 /// </param>
352 /// <returns>
353 /// <para>The link</para>
354 /// </returns>
355 [MethodImpl(MethodImplOptions.AggressiveInlining)]
356 public TLink CountUsages(TLink link)
357 {
358     var root = GetTreeRoot();
359     var total = GetSize(root);
360     var totalRightIgnore = Zero;
361     while (!EqualToZero(root))
362     {
363         var @base = GetBasePartValue(root);
364         if (LessOrEqualThan(@base, link))
365         {
366             root = GetRightOrDefault(root);
367         }
368         else
369         {
370             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
371             root = GetLeftOrDefault(root);
372         }
373     }
374     root = GetTreeRoot();
375     var totalLeftIgnore = Zero;
376     while (!EqualToZero(root))
377     {
378         var @base = GetBasePartValue(root);
379         if (GreaterOrEqualThan(@base, link))
380         {
381             root = GetLeftOrDefault(root);
382         }
383         else
384         {
385             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
386             root = GetRightOrDefault(root);
387         }
388     }
389     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
390 }
391
392 /// <summary>
393 /// <para>
394 /// Eaches the usage using the specified base.
395 /// </para>
396 /// <para></para>
397 /// </summary>
398 /// <param name="@base">
399 /// <para>The base.</para>

```



```

400 /// <para></para>
401 /// </param>
402 /// <param name="handler">
403 /// <para>The handler.</para>
404 /// <para></para>
405 /// </param>
406 /// <returns>
407 /// <para>The link</para>
408 /// <para></para>
409 /// </returns>
410 [MethodImpl(MethodImplOptions.AggressiveInlining)]
411 public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
    ↪ GetTreeRoot(), handler);
412
413 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↪ low-level MSIL stack.
414 [MethodImpl(MethodImplOptions.AggressiveInlining)]
415 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
416 {
417     var @continue = Continue;
418     if (EqualToZero(link))
419     {
420         return @continue;
421     }
422     var linkBasePart = GetBasePartValue(link);
423     var @break = Break;
424     if (GreaterThan(linkBasePart, @base))
425     {
426         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
427         {
428             return @break;
429         }
430     }
431     else if (LessThan(linkBasePart, @base))
432     {
433         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
434         {
435             return @break;
436         }
437     }
438     else //if (linkBasePart == @base)
439     {
440         if (AreEqual(handler(GetLinkValues(link)), @break))
441         {
442             return @break;
443         }
444         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
445         {
446             return @break;
447         }
448         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
449         {
450             return @break;
451         }
452     }
453     return @continue;
454 }
455
456 /// <summary>
457 /// <para>
458 /// Prints the node value using the specified node.
459 /// </para>
460 /// <para></para>
461 /// </summary>
462 /// <param name="node">
463 /// <para>The node.</para>
464 /// <para></para>
465 /// </param>
466 /// <param name="sb">
467 /// <para>The sb.</para>
468 /// <para></para>
469 /// </param>
470 [MethodImpl(MethodImplOptions.AggressiveInlining)]
471 protected override void PrintNodeValue(TLink node, StringBuilder sb)
472 {
473     ref var link = ref GetLinkReference(node);
474     sb.Append(' ');
475     sb.Append(link.Source);

```

```

476         sb.Append('-');
477         sb.Append('>');
478         sb.Append(link.Target);
479     }
480 }
481 }

```

1.83 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links sources avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class LinksSourcesAvlBalancedTreeMethods<TLink> :
15         ↳ LinksAvlBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksSourcesAvlBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public LinksSourcesAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
37             ↳ byte* header) : base(constants, links, header) { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref link</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref TLink GetLeftReference(TLink node) => ref
55             ↳ GetLinkReference(node).LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// <para></para>
70         /// </returns>

```

```

68 [MethodImpl(MethodImplOptions.AggressiveInlining)]
69 protected override ref TLink GetRightReference(TLink node) => ref
    ↪ GetLinkReference(node).RightAsSource;
70
71 /// <summary>
72 /// <para>
73 /// Gets the left using the specified node.
74 /// </para>
75 /// <para></para>
76 /// </summary>
77 /// <param name="node">
78 /// <para>The node.</para>
79 /// <para></para>
80 /// </param>
81 /// <returns>
82 /// <para>The link</para>
83 /// <para></para>
84 /// </returns>
85 [MethodImpl(MethodImplOptions.AggressiveInlining)]
86 protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
87
88 /// <summary>
89 /// <para>
90 /// Gets the right using the specified node.
91 /// </para>
92 /// <para></para>
93 /// </summary>
94 /// <param name="node">
95 /// <para>The node.</para>
96 /// <para></para>
97 /// </param>
98 /// <returns>
99 /// <para>The link</para>
100 /// <para></para>
101 /// </returns>
102 [MethodImpl(MethodImplOptions.AggressiveInlining)]
103 protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
104
105 /// <summary>
106 /// <para>
107 /// Sets the left using the specified node.
108 /// </para>
109 /// <para></para>
110 /// </summary>
111 /// <param name="node">
112 /// <para>The node.</para>
113 /// <para></para>
114 /// </param>
115 /// <param name="left">
116 /// <para>The left.</para>
117 /// <para></para>
118 /// </param>
119 [MethodImpl(MethodImplOptions.AggressiveInlining)]
120 protected override void SetLeft(TLink node, TLink left) =>
    ↪ GetLinkReference(node).LeftAsSource = left;
121
122 /// <summary>
123 /// <para>
124 /// Sets the right using the specified node.
125 /// </para>
126 /// <para></para>
127 /// </summary>
128 /// <param name="node">
129 /// <para>The node.</para>
130 /// <para></para>
131 /// </param>
132 /// <param name="right">
133 /// <para>The right.</para>
134 /// <para></para>
135 /// </param>
136 [MethodImpl(MethodImplOptions.AggressiveInlining)]
137 protected override void SetRight(TLink node, TLink right) =>
    ↪ GetLinkReference(node).RightAsSource = right;
138
139 /// <summary>
140 /// <para>
141 /// Gets the size using the specified node.
142 /// </para>

```

```

143     /// <para></para>
144     /// </summary>
145     /// <param name="node">
146     /// <para>The node.</para>
147     /// <para></para>
148     /// </param>
149     /// <returns>
150     /// <para>The link</para>
151     /// <para></para>
152     /// </returns>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override TLink GetSize(TLink node) =>
155         ↪ GetSizeValue(GetLinkReference(node).SizeAsSource);
156
157     /// <summary>
158     /// <para>
159     /// Sets the size using the specified node.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     /// <param name="node">
164     /// <para>The node.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="size">
168     /// <para>The size.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
173         ↪ GetLinkReference(node).SizeAsSource, size);
174
175     /// <summary>
176     /// <para>
177     /// Determines whether this instance get left is child.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <param name="node">
182     /// <para>The node.</para>
183     /// <para></para>
184     /// </param>
185     /// <returns>
186     /// <para>The bool</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override bool GetLeftIsChild(TLink node) =>
191         ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsSource);
192
193     /// <summary>
194     /// <para>
195     /// Sets the left is child using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <param name="value">
204     /// <para>The value.</para>
205     /// <para></para>
206     /// </param>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override void SetLeftIsChild(TLink node, bool value) =>
209         ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
210
211     /// <summary>
212     /// <para>
213     /// Determines whether this instance get right is child.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="node">
218     /// <para>The node.</para>
219     /// <para></para>
220     /// </param>

```

```

217     /// <returns>
218     /// <para>The bool</para>
219     /// <para></para>
220     /// </returns>
221     [MethodImpl(MethodImplOptions.AggressiveInlining)]
222     protected override bool GetRightIsChild(TLink node) =>
223         ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsSource);
224
225     /// <summary>
226     /// <para>
227     /// Sets the right is child using the specified node.
228     /// </para>
229     /// <para></para>
230     /// </summary>
231     /// <param name="node">
232     /// <para>The node.</para>
233     /// <para></para>
234     /// </param>
235     /// <param name="value">
236     /// <para>The value.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void SetRightIsChild(TLink node, bool value) =>
241         ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
242
243     /// <summary>
244     /// <para>
245     /// Gets the balance using the specified node.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="node">
250     /// <para>The node.</para>
251     /// <para></para>
252     /// </param>
253     /// <returns>
254     /// <para>The sbyte</para>
255     /// <para></para>
256     /// </returns>
257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
258     protected override sbyte GetBalance(TLink node) =>
259         ↪ GetBalanceValue(GetLinkReference(node).SizeAsSource);
260
261     /// <summary>
262     /// <para>
263     /// Sets the balance using the specified node.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="node">
268     /// <para>The node.</para>
269     /// <para></para>
270     /// </param>
271     /// <param name="value">
272     /// <para>The value.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
277         ↪ GetLinkReference(node).SizeAsSource, value);
278
279     /// <summary>
280     /// <para>
281     /// Gets the tree root.
282     /// </para>
283     /// <para></para>
284     /// </summary>
285     /// <returns>
286     /// <para>The link</para>
287     /// <para></para>
288     /// </returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
291
292     /// <summary>
293     /// <para>
294     /// Gets the base part value using the specified link.

```

```

291    /// </para>
292    /// <para></para>
293    /// </summary>
294    /// <param name="link">
295    /// <para>The link.</para>
296    /// <para></para>
297    /// </param>
298    /// <returns>
299    /// <para>The link</para>
300    /// <para></para>
301    /// </returns>
302    [MethodImpl(MethodImplOptions.AggressiveInlining)]
303    protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
304
305    /// <summary>
306    /// <para>
307    /// Determines whether this instance first is to the left of second.
308    /// </para>
309    /// <para></para>
310    /// </summary>
311    /// <param name="firstSource">
312    /// <para>The first source.</para>
313    /// <para></para>
314    /// </param>
315    /// <param name="firstTarget">
316    /// <para>The first target.</para>
317    /// <para></para>
318    /// </param>
319    /// <param name="secondSource">
320    /// <para>The second source.</para>
321    /// <para></para>
322    /// </param>
323    /// <param name="secondTarget">
324    /// <para>The second target.</para>
325    /// <para></para>
326    /// </param>
327    /// <returns>
328    /// <para>The bool</para>
329    /// <para></para>
330    /// </returns>
331    [MethodImpl(MethodImplOptions.AggressiveInlining)]
332    protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
333    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
334    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the right of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="firstSource">
343    /// <para>The first source.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="firstTarget">
347    /// <para>The first target.</para>
348    /// <para></para>
349    /// </param>
350    /// <param name="secondSource">
351    /// <para>The second source.</para>
352    /// <para></para>
353    /// </param>
354    /// <param name="secondTarget">
355    /// <para>The second target.</para>
356    /// <para></para>
357    /// </param>
358    /// <returns>
359    /// <para>The bool</para>
360    /// <para></para>
361    /// </returns>
362    [MethodImpl(MethodImplOptions.AggressiveInlining)]
363    protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
364    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
365    ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
366
367    /// <summary>

```

```

364     /// <para>
365     /// Clears the node using the specified node.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="node">
370     /// <para>The node.</para>
371     /// <para></para>
372     /// </param>
373     [MethodImpl(MethodImplOptions.AggressiveInlining)]
374     protected override void ClearNode(TLink node)
375     {
376         ref var link = ref GetLinkReference(node);
377         link.LeftAsSource = Zero;
378         link.RightAsSource = Zero;
379         link.SizeAsSource = Zero;
380     }
381 }
382 }

```

1.84 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMeth

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class LinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
15        ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="LinksSourcesRecursionlessSizeBalancedTreeMethods"/>
20        ↳ instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
38            ↳ byte* links, byte* header) : base(constants, links, header) { }
39
40        /// <summary>
41        /// <para>
42        /// Gets the left reference using the specified node.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="node">
47        /// <para>The node.</para>
48        /// <para></para>
49        /// </param>
50        /// <returns>
51        /// <para>The ref link</para>
52        /// <para></para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        protected override ref TLink GetLeftReference(TLink node) => ref
56            ↳ GetLinkReference(node).LeftAsSource;
57    }
58 }

```

```

54    /// <summary>
55    /// <para>
56    /// Gets the right reference using the specified node.
57    /// </para>
58    /// <para></para>
59    /// </summary>
60    /// <param name="node">
61    /// <para>The node.</para>
62    /// <para></para>
63    /// </param>
64    /// <returns>
65    /// <para>The ref link</para>
66    /// <para></para>
67    /// </returns>
68    [MethodImpl(MethodImplOptions.AggressiveInlining)]
69    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ GetLinkReference(node).RightAsSource;
70
71    /// <summary>
72    /// <para>
73    /// Gets the left using the specified node.
74    /// </para>
75    /// <para></para>
76    /// </summary>
77    /// <param name="node">
78    /// <para>The node.</para>
79    /// <para></para>
80    /// </param>
81    /// <returns>
82    /// <para>The link</para>
83    /// <para></para>
84    /// </returns>
85    [MethodImpl(MethodImplOptions.AggressiveInlining)]
86    protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
87
88    /// <summary>
89    /// <para>
90    /// Gets the right using the specified node.
91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="node">
95    /// <para>The node.</para>
96    /// <para></para>
97    /// </param>
98    /// <returns>
99    /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
104
105    /// <summary>
106    /// <para>
107    /// Sets the left using the specified node.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="node">
112    /// <para>The node.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="left">
116    /// <para>The left.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override void SetLeft(TLink node, TLink left) =>
    ↪ GetLinkReference(node).LeftAsSource = left;
121
122    /// <summary>
123    /// <para>
124    /// Sets the right using the specified node.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="node">
129    /// <para>The node.</para>

```



```

130    /// <para></para>
131    /// </param>
132    /// <param name="right">
133    /// <para>The right.</para>
134    /// <para></para>
135    /// </param>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    protected override void SetRight(TLink node, TLink right) =>
138        ↪ GetLinkReference(node).RightAsSource = right;
139
140    /// <summary>
141    /// <para>
142    /// Gets the size using the specified node.
143    /// </para>
144    /// <para></para>
145    /// </summary>
146    /// <param name="node">
147    /// <para>The node.</para>
148    /// <para></para>
149    /// </param>
150    /// <returns>
151    /// <para>The link</para>
152    /// <para></para>
153    /// </returns>
154    [MethodImpl(MethodImplOptions.AggressiveInlining)]
155    protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsSource;
156
157    /// <summary>
158    /// <para>
159    /// Sets the size using the specified node.
160    /// </para>
161    /// <para></para>
162    /// </summary>
163    /// <param name="node">
164    /// <para>The node.</para>
165    /// <para></para>
166    /// </param>
167    /// <param name="size">
168    /// <para>The size.</para>
169    /// <para></para>
170    /// </param>
171    [MethodImpl(MethodImplOptions.AggressiveInlining)]
172    protected override void SetSize(TLink node, TLink size) =>
173        ↪ GetLinkReference(node).SizeAsSource = size;
174
175    /// <summary>
176    /// <para>
177    /// Gets the tree root.
178    /// </para>
179    /// <para></para>
180    /// </summary>
181    /// <returns>
182    /// <para>The link</para>
183    /// <para></para>
184    /// </returns>
185    [MethodImpl(MethodImplOptions.AggressiveInlining)]
186    protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
187
188    /// <summary>
189    /// <para>
190    /// Gets the base part value using the specified link.
191    /// </para>
192    /// <para></para>
193    /// </summary>
194    /// <param name="link">
195    /// <para>The link.</para>
196    /// <para></para>
197    /// </param>
198    /// <returns>
199    /// <para>The link</para>
200    /// <para></para>
201    /// </returns>
202    [MethodImpl(MethodImplOptions.AggressiveInlining)]
203    protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
204
205    /// <summary>
206    /// <para>
207    /// Determines whether this instance first is to the left of second.

```

```

206    /// </para>
207    /// <para></para>
208    /// </summary>
209    /// <param name="firstSource">
210    /// <para>The first source.</para>
211    /// <para></para>
212    /// </param>
213    /// <param name="firstTarget">
214    /// <para>The first target.</para>
215    /// <para></para>
216    /// </param>
217    /// <param name="secondSource">
218    /// <para>The second source.</para>
219    /// <para></para>
220    /// </param>
221    /// <param name="secondTarget">
222    /// <para>The second target.</para>
223    /// <para></para>
224    /// </param>
225    /// <returns>
226    /// <para>The bool</para>
227    /// <para></para>
228    /// </returns>
229    [MethodImpl(MethodImplOptions.AggressiveInlining)]
230    protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
231
232    /// <summary>
233    /// <para>
234    /// Determines whether this instance first is to the right of second.
235    /// </para>
236    /// <para></para>
237    /// </summary>
238    /// <param name="firstSource">
239    /// <para>The first source.</para>
240    /// <para></para>
241    /// </param>
242    /// <param name="firstTarget">
243    /// <para>The first target.</para>
244    /// <para></para>
245    /// </param>
246    /// <param name="secondSource">
247    /// <para>The second source.</para>
248    /// <para></para>
249    /// </param>
250    /// <param name="secondTarget">
251    /// <para>The second target.</para>
252    /// <para></para>
253    /// </param>
254    /// <returns>
255    /// <para>The bool</para>
256    /// <para></para>
257    /// </returns>
258    [MethodImpl(MethodImplOptions.AggressiveInlining)]
259    protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
260
261    /// <summary>
262    /// <para>
263    /// Clears the node using the specified node.
264    /// </para>
265    /// <para></para>
266    /// </summary>
267    /// <param name="node">
268    /// <para>The node.</para>
269    /// <para></para>
270    /// </param>
271    [MethodImpl(MethodImplOptions.AggressiveInlining)]
272    protected override void ClearNode(TLink node)
273    {
274        ref var link = ref GetLinkReference(node);
275        link.LeftAsSource = Zero;
276        link.RightAsSource = Zero;
277        link.SizeAsSource = Zero;
278    }
279    }

```

1.85 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links sources size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{TLink}" />
14     public unsafe class LinksSourcesSizeBalancedTreeMethods<TLink> :
15         ↳ LinksSizeBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods" /> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public LinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
37             ↳ byte* header) : base(constants, links, header) { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref link</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref TLink GetLeftReference(TLink node) => ref
55             ↳ GetLinkReference(node).LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref TLink GetRightReference(TLink node) => ref
73             ↳ GetLinkReference(node).RightAsSource;
74
75         /// <summary>

```

```

72     /// <para>
73     /// Gets the left using the specified node.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
87
88     /// <summary>
89     /// <para>
90     /// Gets the right using the specified node.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
104
105    /// <summary>
106    /// <para>
107    /// Sets the left using the specified node.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="node">
112    /// <para>The node.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="left">
116    /// <para>The left.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override void SetLeft(TLink node, TLink left) =>
121        ↪ GetLinkReference(node).LeftAsSource = left;
122
123    /// <summary>
124    /// <para>
125    /// Sets the right using the specified node.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="node">
130    /// <para>The node.</para>
131    /// <para></para>
132    /// </param>
133    /// <param name="right">
134    /// <para>The right.</para>
135    /// <para></para>
136    /// </param>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    protected override void SetRight(TLink node, TLink right) =>
139        ↪ GetLinkReference(node).RightAsSource = right;
140
141    /// <summary>
142    /// <para>
143    /// Gets the size using the specified node.
144    /// </para>
145    /// <para></para>
146    /// </summary>
147    /// <param name="node">
148    /// <para>The node.</para>
149    /// <para></para>

```

```

148     /// </param>
149     /// <returns>
150     /// <para>The link</para>
151     /// <para></para>
152     /// </returns>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsSource;
155
156     /// <summary>
157     /// <para>
158     /// Sets the size using the specified node.
159     /// </para>
160     /// <para></para>
161     /// </summary>
162     /// <param name="node">
163     /// <para>The node.</para>
164     /// <para></para>
165     /// </param>
166     /// <param name="size">
167     /// <para>The size.</para>
168     /// <para></para>
169     /// </param>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override void SetSize(TLink node, TLink size) =>
172     ↪ GetLinkReference(node).SizeAsSource = size;
173
174     /// <summary>
175     /// <para>
176     /// Gets the tree root.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <returns>
181     /// <para>The link</para>
182     /// <para></para>
183     /// </returns>
184     [MethodImpl(MethodImplOptions.AggressiveInlining)]
185     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
186
187     /// <summary>
188     /// <para>
189     /// Gets the base part value using the specified link.
190     /// </para>
191     /// <para></para>
192     /// </summary>
193     /// <param name="link">
194     /// <para>The link.</para>
195     /// <para></para>
196     /// </param>
197     /// <returns>
198     /// <para>The link</para>
199     /// <para></para>
200     /// </returns>
201     [MethodImpl(MethodImplOptions.AggressiveInlining)]
202     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
203
204     /// <summary>
205     /// <para>
206     /// Determines whether this instance first is to the left of second.
207     /// </para>
208     /// <para></para>
209     /// </summary>
210     /// <param name="firstSource">
211     /// <para>The first source.</para>
212     /// <para></para>
213     /// </param>
214     /// <param name="firstTarget">
215     /// <para>The first target.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="secondSource">
219     /// <para>The second source.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondTarget">
223     /// <para>The second target.</para>
224     /// <para></para>
225     /// </param>

```

```

225     /// <returns>
226     /// <para>The bool</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));

231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));

260
261     /// <summary>
262     /// <para>
263     /// Clears the node using the specified node.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="node">
268     /// <para>The node.</para>
269     /// <para></para>
270     /// </param>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     protected override void ClearNode(TLink node)
273     {
274         ref var link = ref GetLinkReference(node);
275         link.LeftAsSource = Zero;
276         link.RightAsSource = Zero;
277         link.SizeAsSource = Zero;
278     }
279 }
280 }

```

1.86 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links targets avl balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksAvlBalancedTreeMethodsBase{TLink}" />
14    public unsafe class LinksTargetsAvlBalancedTreeMethods<TLink> :
    ↪ LinksAvlBalancedTreeMethodsBase<TLink>
15    {

```

```

16    /// <summary>
17    /// <para>
18    /// Initializes a new <see cref="LinksTargetsAvlBalancedTreeMethods"/> instance.
19    /// </para>
20    /// <para></para>
21    /// </summary>
22    /// <param name="constants">
23    /// <para>A constants.</para>
24    /// <para></para>
25    /// </param>
26    /// <param name="links">
27    /// <para>A links.</para>
28    /// <para></para>
29    /// </param>
30    /// <param name="header">
31    /// <para>A header.</para>
32    /// <para></para>
33    /// </param>
34    [MethodImpl(MethodImplOptions.AggressiveInlining)]
35    public LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
36    ↪ byte* header) : base(constants, links, header) { }
37
38    /// <summary>
39    /// <para>
40    /// Gets the left reference using the specified node.
41    /// </para>
42    /// <para></para>
43    /// </summary>
44    /// <param name="node">
45    /// <para>The node.</para>
46    /// <para></para>
47    /// </param>
48    /// <returns>
49    /// <para>The ref link</para>
50    /// <para></para>
51    /// </returns>
52    [MethodImpl(MethodImplOptions.AggressiveInlining)]
53    protected override ref TLink GetLeftReference(TLink node) => ref
54    ↪ GetLinkReference(node).LeftAsTarget;
55
56    /// <summary>
57    /// <para>
58    /// Gets the right reference using the specified node.
59    /// </para>
60    /// <para></para>
61    /// </summary>
62    /// <param name="node">
63    /// <para>The node.</para>
64    /// <para></para>
65    /// </param>
66    /// <returns>
67    /// <para>The ref link</para>
68    /// <para></para>
69    /// </returns>
70    [MethodImpl(MethodImplOptions.AggressiveInlining)]
71    protected override ref TLink GetRightReference(TLink node) => ref
72    ↪ GetLinkReference(node).RightAsTarget;
73
74    /// <summary>
75    /// <para>
76    /// Gets the left using the specified node.
77    /// </para>
78    /// <para></para>
79    /// </summary>
80    /// <param name="node">
81    /// <para>The node.</para>
82    /// <para></para>
83    /// </param>
84    /// <returns>
85    /// <para>The link</para>
86    /// <para></para>
87    /// </returns>
88    [MethodImpl(MethodImplOptions.AggressiveInlining)]
89    protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsTarget;
90
91    /// <summary>
92    /// <para>
93    /// Gets the right using the specified node.

```

```

91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="node">
95    /// <para>The node.</para>
96    /// <para></para>
97    /// </param>
98    /// <returns>
99    /// <para>The link</para>
100   /// <para></para>
101   /// </returns>
102   [MethodImpl(MethodImplOptions.AggressiveInlining)]
103   protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;
104
105   /// <summary>
106   /// <para>
107   /// Sets the left using the specified node.
108   /// </para>
109   /// <para></para>
110   /// </summary>
111   /// <param name="node">
112   /// <para>The node.</para>
113   /// <para></para>
114   /// </param>
115   /// <param name="left">
116   /// <para>The left.</para>
117   /// <para></para>
118   /// </param>
119   [MethodImpl(MethodImplOptions.AggressiveInlining)]
120   protected override void SetLeft(TLink node, TLink left) =>
121   ↪ GetLinkReference(node).LeftAsTarget = left;
122
123   /// <summary>
124   /// <para>
125   /// Sets the right using the specified node.
126   /// </para>
127   /// <para></para>
128   /// </summary>
129   /// <param name="node">
130   /// <para>The node.</para>
131   /// <para></para>
132   /// </param>
133   /// <param name="right">
134   /// <para>The right.</para>
135   /// <para></para>
136   /// </param>
137   [MethodImpl(MethodImplOptions.AggressiveInlining)]
138   protected override void SetRight(TLink node, TLink right) =>
139   ↪ GetLinkReference(node).RightAsTarget = right;
140
141   /// <summary>
142   /// <para>
143   /// Gets the size using the specified node.
144   /// </para>
145   /// <para></para>
146   /// </summary>
147   /// <param name="node">
148   /// <para>The node.</para>
149   /// <para></para>
150   /// </param>
151   /// <returns>
152   /// <para>The link</para>
153   /// <para></para>
154   /// </returns>
155   [MethodImpl(MethodImplOptions.AggressiveInlining)]
156   protected override TLink GetSize(TLink node) =>
157   ↪ GetSizeValue(GetLinkReference(node).SizeAsTarget);
158
159   /// <summary>
160   /// <para>
161   /// Sets the size using the specified node.
162   /// </para>
163   /// <para></para>
164   /// </summary>
165   /// <param name="node">
166   /// <para>The node.</para>
167   /// <para></para>
168   /// </param>

```



```

166    /// <param name="size">
167    /// <para>The size.</para>
168    /// <para></para>
169    /// </param>
170    [MethodImpl(MethodImplOptions.AggressiveInlining)]
171    protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
    ↳ GetLinkReference(node).SizeAsTarget, size);

172
173    /// <summary>
174    /// <para>
175    /// Determines whether this instance get left is child.
176    /// </para>
177    /// <para></para>
178    /// </summary>
179    /// <param name="node">
180    /// <para>The node.</para>
181    /// <para></para>
182    /// </param>
183    /// <returns>
184    /// <para>The bool</para>
185    /// <para></para>
186    /// </returns>
187    [MethodImpl(MethodImplOptions.AggressiveInlining)]
188    protected override bool GetLeftIsChild(TLink node) =>
    ↳ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);

189
190    /// <summary>
191    /// <para>
192    /// Sets the left is child using the specified node.
193    /// </para>
194    /// <para></para>
195    /// </summary>
196    /// <param name="node">
197    /// <para>The node.</para>
198    /// <para></para>
199    /// </param>
200    /// <param name="value">
201    /// <para>The value.</para>
202    /// <para></para>
203    /// </param>
204    [MethodImpl(MethodImplOptions.AggressiveInlining)]
205    protected override void SetLeftIsChild(TLink node, bool value) =>
    ↳ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);

206
207    /// <summary>
208    /// <para>
209    /// Determines whether this instance get right is child.
210    /// </para>
211    /// <para></para>
212    /// </summary>
213    /// <param name="node">
214    /// <para>The node.</para>
215    /// <para></para>
216    /// </param>
217    /// <returns>
218    /// <para>The bool</para>
219    /// <para></para>
220    /// </returns>
221    [MethodImpl(MethodImplOptions.AggressiveInlining)]
222    protected override bool GetRightIsChild(TLink node) =>
    ↳ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);

223
224    /// <summary>
225    /// <para>
226    /// Sets the right is child using the specified node.
227    /// </para>
228    /// <para></para>
229    /// </summary>
230    /// <param name="node">
231    /// <para>The node.</para>
232    /// <para></para>
233    /// </param>
234    /// <param name="value">
235    /// <para>The value.</para>
236    /// <para></para>
237    /// </param>
238    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

239     protected override void SetRightIsChild(TLink node, bool value) =>
240         ↳ SetRightIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
241
242     /// <summary>
243     /// <para>
244     /// Gets the balance using the specified node.
245     /// </para>
246     /// </summary>
247     /// <param name="node">
248     /// <para>The node.</para>
249     /// </param>
250     /// </returns>
251     /// <para>The sbyte</para>
252     /// </returns>
253     [MethodImpl(MethodImplOptions.AggressiveInlining)]
254     protected override sbyte GetBalance(TLink node) =>
255         ↳ GetBalanceValue(GetLinkReference(node).SizeAsTarget);
256
257     /// <summary>
258     /// <para>
259     /// Sets the balance using the specified node.
260     /// </para>
261     /// </summary>
262     /// <param name="node">
263     /// <para>The node.</para>
264     /// </param>
265     /// <param name="value">
266     /// <para>The value.</para>
267     /// </param>
268     /// </returns>
269     [MethodImpl(MethodImplOptions.AggressiveInlining)]
270     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
271         ↳ GetLinkReference(node).SizeAsTarget, value);
272
273     /// <summary>
274     /// <para>
275     /// Gets the tree root.
276     /// </para>
277     /// </summary>
278     /// </returns>
279     /// <para>The link</para>
280     /// </returns>
281     [MethodImpl(MethodImplOptions.AggressiveInlining)]
282     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
283
284     /// <summary>
285     /// <para>
286     /// Gets the base part value using the specified link.
287     /// </para>
288     /// </summary>
289     /// <param name="link">
290     /// <para>The link.</para>
291     /// </param>
292     /// </returns>
293     /// <para>The link</para>
294     /// </returns>
295     [MethodImpl(MethodImplOptions.AggressiveInlining)]
296     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
297
298     /// <summary>
299     /// <para>
300     /// Determines whether this instance first is to the left of second.
301     /// </para>
302     /// </summary>
303     /// <param name="firstSource">
304     /// <para>The first source.</para>
305     /// </param>
306     /// </returns>
307

```

```

314     /// </param>
315     /// <param name="firstTarget">
316     /// <para>The first target.</para>
317     /// <para></para>
318     /// </param>
319     /// <param name="secondSource">
320     /// <para>The second source.</para>
321     /// <para></para>
322     /// </param>
323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// <para></para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));

333     /// <summary>
334     /// <para>
335     /// Determines whether this instance first is to the right of second.
336     /// </para>
337     /// <para></para>
338     /// </summary>
339     /// <param name="firstSource">
340     /// <para>The first source.</para>
341     /// <para></para>
342     /// </param>
343     /// <param name="firstTarget">
344     /// <para>The first target.</para>
345     /// <para></para>
346     /// </param>
347     /// <param name="secondSource">
348     /// <para>The second source.</para>
349     /// <para></para>
350     /// </param>
351     /// <param name="secondTarget">
352     /// <para>The second target.</para>
353     /// <para></para>
354     /// </param>
355     /// <returns>
356     /// <para>The bool</para>
357     /// <para></para>
358     /// </returns>
359     [MethodImpl(MethodImplOptions.AggressiveInlining)]
360     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

362     /// <summary>
363     /// <para>
364     /// Clears the node using the specified node.
365     /// </para>
366     /// <para></para>
367     /// </summary>
368     /// <param name="node">
369     /// <para>The node.</para>
370     /// <para></para>
371     /// </param>
372     [MethodImpl(MethodImplOptions.AggressiveInlining)]
373     protected override void ClearNode(TLink node)
374     {
375         {
376             ref var link = ref GetLinkReference(node);
377             link.LeftAsTarget = Zero;
378             link.RightAsTarget = Zero;
379             link.SizeAsTarget = Zero;
380         }
381     }
382 }

```

1.87 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethod

using System.Runtime.CompilerServices;

2

3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

```

4
5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
15        ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
20        ↳ instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
38            ↳ byte* links, byte* header) : base(constants, links, header) { }
39
40        /// <summary>
41        /// <para>
42        /// Gets the left reference using the specified node.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="node">
47        /// <para>The node.</para>
48        /// <para></para>
49        /// </param>
50        /// <returns>
51        /// <para>The ref link</para>
52        /// <para></para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        protected override ref TLink GetLeftReference(TLink node) => ref
56            ↳ GetLinkReference(node).LeftAsTarget;
57
58        /// <summary>
59        /// <para>
60        /// Gets the right reference using the specified node.
61        /// </para>
62        /// <para></para>
63        /// </summary>
64        /// <param name="node">
65        /// <para>The node.</para>
66        /// <para></para>
67        /// </param>
68        /// <returns>
69        /// <para>The ref link</para>
70        /// <para></para>
71        /// </returns>
72        [MethodImpl(MethodImplOptions.AggressiveInlining)]
73        protected override ref TLink GetRightReference(TLink node) => ref
74            ↳ GetLinkReference(node).RightAsTarget;
75
76        /// <summary>
77        /// <para>
78        /// Gets the left using the specified node.
79        /// </para>
80        /// <para></para>
81        /// </summary>

```

```

77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsTarget;
87
88     /// <summary>
89     /// <para>
90     /// Gets the right using the specified node.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;
104
105    /// <summary>
106    /// <para>
107    /// Sets the left using the specified node.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="node">
112    /// <para>The node.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="left">
116    /// <para>The left.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override void SetLeft(TLink node, TLink left) =>
121        ↪ GetLinkReference(node).LeftAsTarget = left;
122
123    /// <summary>
124    /// <para>
125    /// Sets the right using the specified node.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="node">
130    /// <para>The node.</para>
131    /// <para></para>
132    /// </param>
133    /// <param name="right">
134    /// <para>The right.</para>
135    /// <para></para>
136    /// </param>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    protected override void SetRight(TLink node, TLink right) =>
139        ↪ GetLinkReference(node).RightAsTarget = right;
140
141    /// <summary>
142    /// <para>
143    /// Gets the size using the specified node.
144    /// </para>
145    /// <para></para>
146    /// </summary>
147    /// <param name="node">
148    /// <para>The node.</para>
149    /// <para></para>
150    /// </param>
151    /// <returns>
152    /// <para>The link</para>
153    /// <para></para>
154    /// </returns>

```

```

153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsTarget;
155
156 /// <summary>
157 /// <para>
158 /// Sets the size using the specified node.
159 /// </para>
160 /// <para></para>
161 /// </summary>
162 /// <param name="node">
163 /// <para>The node.</para>
164 /// <para></para>
165 /// </param>
166 /// <param name="size">
167 /// <para>The size.</para>
168 /// <para></para>
169 /// </param>
170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 protected override void SetSize(TLink node, TLink size) =>
172     ↪ GetLinkReference(node).SizeAsTarget = size;
173
174 /// <summary>
175 /// <para>
176 /// Gets the tree root.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <returns>
181 /// <para>The link</para>
182 /// <para></para>
183 /// </returns>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
186
187 /// <summary>
188 /// <para>
189 /// Gets the base part value using the specified link.
190 /// </para>
191 /// <para></para>
192 /// </summary>
193 /// <param name="link">
194 /// <para>The link.</para>
195 /// <para></para>
196 /// </param>
197 /// <returns>
198 /// <para>The link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
203
204 /// <summary>
205 /// <para>
206 /// Determines whether this instance first is to the left of second.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <param name="firstSource">
211 /// <para>The first source.</para>
212 /// <para></para>
213 /// </param>
214 /// <param name="firstTarget">
215 /// <para>The first target.</para>
216 /// <para></para>
217 /// </param>
218 /// <param name="secondSource">
219 /// <para>The second source.</para>
220 /// <para></para>
221 /// </param>
222 /// <param name="secondTarget">
223 /// <para>The second target.</para>
224 /// <para></para>
225 /// </param>
226 /// <returns>
227 /// <para>The bool</para>
228 /// <para></para>
229 /// </returns>
230 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

230     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
260
261     /// <summary>
262     /// <para>
263     /// Clears the node using the specified node.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="node">
268     /// <para>The node.</para>
269     /// <para></para>
270     /// </param>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     protected override void ClearNode(TLink node)
273     {
274         ref var link = ref GetLinkReference(node);
275         link.LeftAsTarget = Zero;
276         link.RightAsTarget = Zero;
277         link.SizeAsTarget = Zero;
278     }
279 }
280 }

```

1.88 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{TLink}" />
14     public unsafe class LinksTargetsSizeBalancedTreeMethods<TLink> :
    ↪ LinksSizeBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksTargetsSizeBalancedTreeMethods" /> instance.
19         /// </para>
20         /// <para></para>

```

```

21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public LinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
36         ↳ byte* header) : base(constants, links, header) { }
37
38     /// <summary>
39     /// <para>
40     /// Gets the left reference using the specified node.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     /// <param name="node">
45     /// <para>The node.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The ref link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override ref TLink GetLeftReference(TLink node) => ref
54     ↳ GetLinkReference(node).LeftAsTarget;
55
56     /// <summary>
57     /// <para>
58     /// Gets the right reference using the specified node.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="node">
63     /// <para>The node.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The ref link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref TLink GetRightReference(TLink node) => ref
72     ↳ GetLinkReference(node).RightAsTarget;
73
74     /// <summary>
75     /// <para>
76     /// Gets the left using the specified node.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="node">
81     /// <para>The node.</para>
82     /// <para></para>
83     /// </param>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsTarget;
90
91     /// <summary>
92     /// <para>
93     /// Gets the right using the specified node.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <param name="node">
98     /// <para>The node.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The link</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;

```



```

96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;
104
105    /// <summary>
106    /// <para>
107    /// Sets the left using the specified node.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="node">
112    /// <para>The node.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="left">
116    /// <para>The left.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override void SetLeft(TLink node, TLink left) =>
121        ↪ GetLinkReference(node).LeftAsTarget = left;
122
123    /// <summary>
124    /// <para>
125    /// Sets the right using the specified node.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="node">
130    /// <para>The node.</para>
131    /// <para></para>
132    /// </param>
133    /// <param name="right">
134    /// <para>The right.</para>
135    /// <para></para>
136    /// </param>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    protected override void SetRight(TLink node, TLink right) =>
139        ↪ GetLinkReference(node).RightAsTarget = right;
140
141    /// <summary>
142    /// <para>
143    /// Gets the size using the specified node.
144    /// </para>
145    /// <para></para>
146    /// </summary>
147    /// <param name="node">
148    /// <para>The node.</para>
149    /// <para></para>
150    /// </param>
151    /// <returns>
152    /// <para>The link</para>
153    /// <para></para>
154    /// </returns>
155    [MethodImpl(MethodImplOptions.AggressiveInlining)]
156    protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsTarget;
157
158    /// <summary>
159    /// <para>
160    /// Sets the size using the specified node.
161    /// </para>
162    /// <para></para>
163    /// </summary>
164    /// <param name="node">
165    /// <para>The node.</para>
166    /// <para></para>
167    /// </param>
168    /// <param name="size">
169    /// <para>The size.</para>
170    /// <para></para>
171    /// </param>
172    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

171 protected override void SetSize(TLink node, TLink size) =>
172     ↳ GetLinkReference(node).SizeAsTarget = size;
173
174 /// <summary>
175 /// <para>
176 /// Gets the tree root.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <returns>
181 /// <para>The link</para>
182 /// <para></para>
183 /// </returns>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
186
187 /// <summary>
188 /// <para>
189 /// Gets the base part value using the specified link.
190 /// </para>
191 /// <para></para>
192 /// </summary>
193 /// <param name="link">
194 /// <para>The link.</para>
195 /// <para></para>
196 /// </param>
197 /// <returns>
198 /// <para>The link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
203
204 /// <summary>
205 /// <para>
206 /// Determines whether this instance first is to the left of second.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <param name="firstSource">
211 /// <para>The first source.</para>
212 /// <para></para>
213 /// </param>
214 /// <param name="firstTarget">
215 /// <para>The first target.</para>
216 /// <para></para>
217 /// </param>
218 /// <param name="secondSource">
219 /// <para>The second source.</para>
220 /// <para></para>
221 /// </param>
222 /// <param name="secondTarget">
223 /// <para>The second target.</para>
224 /// <para></para>
225 /// </param>
226 /// <returns>
227 /// <para>The bool</para>
228 /// <para></para>
229 /// </returns>
230 [MethodImpl(MethodImplOptions.AggressiveInlining)]
231 protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
232     ↳ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
233     ↳ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
234
235 /// <summary>
236 /// <para>
237 /// Determines whether this instance first is to the right of second.
238 /// </para>
239 /// <para></para>
240 /// </summary>
241 /// <param name="firstSource">
242 /// <para>The first source.</para>
243 /// <para></para>
244 /// </param>
245 /// <param name="firstTarget">
246 /// <para>The first target.</para>
247 /// <para></para>
248 /// </param>
249 /// <returns>
250 /// <para>The bool</para>
251 /// <para></para>
252 /// </returns>

```

```

246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

260
261     /// <summary>
262     /// <para>
263     /// Clears the node using the specified node.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="node">
268     /// <para>The node.</para>
269     /// <para></para>
270     /// </param>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     protected override void ClearNode(TLink node)
273     {
274         ref var link = ref GetLinkReference(node);
275         link.LeftAsTarget = Zero;
276         link.RightAsTarget = Zero;
277         link.SizeAsTarget = Zero;
278     }
279 }
280 }

```

1.89 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Generic
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the united memory links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="UnitedMemoryLinksBase{TLink}" />
18     public unsafe class UnitedMemoryLinks<TLink> : UnitedMemoryLinksBase<TLink>
19     {
20         private readonly Func<ILinksTreeMethods<TLink>> _createSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<TLink>> _createTargetTreeMethods;
22         private byte* _header;
23         private byte* _links;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="UnitedMemoryLinks" /> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="address">
32         /// <para>A address.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
37
38         /// <summary>
39         /// Создаёт экземпляры базы данных Links в файле по указанному адресу, с указанным
        ↪ минимальным шагом расширения базы данных.

```

```

40  /// </summary>
41  /// <param name="address">Полный путь к файлу базы данных.</param>
42  /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
    ↳ байтах.</param>
43  [MethodImpl(MethodImplOptions.AggressiveInlining)]
44  public UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
    ↳ FileMappedResizableDirectMemory(address, memoryReservationStep),
    ↳ memoryReservationStep) { }
45
46  /// <summary>
47  /// <para>
48  /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
49  /// </para>
50  /// <para></para>
51  /// </summary>
52  /// <param name="memory">
53  /// <para>A memory.</para>
54  /// <para></para>
55  /// </param>
56  [MethodImpl(MethodImplOptions.AggressiveInlining)]
57  public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
    ↳ DefaultLinksSizeStep) { }
58
59  /// <summary>
60  /// <para>
61  /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
62  /// </para>
63  /// <para></para>
64  /// </summary>
65  /// <param name="memory">
66  /// <para>A memory.</para>
67  /// <para></para>
68  /// </param>
69  /// <param name="memoryReservationStep">
70  /// <para>A memory reservation step.</para>
71  /// <para></para>
72  /// </param>
73  [MethodImpl(MethodImplOptions.AggressiveInlining)]
74  public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep) :
    ↳ this(memory, memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
    ↳ IndexTreeType.Default) { }
75
76  /// <summary>
77  /// <para>
78  /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
79  /// </para>
80  /// <para></para>
81  /// </summary>
82  /// <param name="memory">
83  /// <para>A memory.</para>
84  /// <para></para>
85  /// </param>
86  /// <param name="memoryReservationStep">
87  /// <para>A memory reservation step.</para>
88  /// <para></para>
89  /// </param>
90  /// <param name="constants">
91  /// <para>A constants.</para>
92  /// <para></para>
93  /// </param>
94  /// <param name="indexTreeType">
95  /// <para>A index tree type.</para>
96  /// <para></para>
97  /// </param>
98  [MethodImpl(MethodImplOptions.AggressiveInlining)]
99  public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep,
    ↳ LinksConstants<TLink> constants, IndexTreeType indexTreeType) : base(memory,
    ↳ memoryReservationStep, constants)
100  {
101      if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
102      {
103          _createSourceTreeMethods = () => new
            ↳ LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
104          _createTargetTreeMethods = () => new
            ↳ LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
105      }
106      else

```

```

107     {
108         _createSourceTreeMethods = () => new
109             ↳ LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
110         _createTargetTreeMethods = () => new
111             ↳ LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
112     }
113     Init(memory, memoryReservationStep);
114 }
115
116 /// <summary>
117 /// <para>
118 /// Sets the pointers using the specified memory.
119 /// </para>
120 /// </summary>
121 /// <param name="memory">
122 /// <para>The memory.</para>
123 /// </param>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 protected override void SetPointers(IResizableDirectMemory memory)
126 {
127     _links = (byte*)memory.Pointer;
128     _header = _links;
129     SourcesTreeMethods = _createSourceTreeMethods();
130     TargetsTreeMethods = _createTargetTreeMethods();
131     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_links, _header);
132 }
133
134 /// <summary>
135 /// <para>
136 /// Resets the pointers.
137 /// </para>
138 /// </summary>
139 [MethodImpl(MethodImplOptions.AggressiveInlining)]
140 protected override void ResetPointers()
141 {
142     base.ResetPointers();
143     _links = null;
144     _header = null;
145 }
146
147 /// <summary>
148 /// <para>
149 /// Gets the header reference.
150 /// </para>
151 /// </summary>
152 /// <returns>
153 /// <para>A ref links header of t link</para>
154 /// </returns>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 protected override ref LinksHeader<TLink> GetHeaderReference() => ref
157     ↳ AsRef<LinksHeader<TLink>>(_header);
158
159 /// <summary>
160 /// <para>
161 /// Gets the link reference using the specified link index.
162 /// </para>
163 /// </summary>
164 /// <param name="linkIndex">
165 /// <para>The link index.</para>
166 /// </param>
167 /// <returns>
168 /// <para>A ref raw link of t link</para>
169 /// </returns>
170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
172     ↳ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * Convert.ToInt64(linkIndex)));
173 }
174 }

```

1.90 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Disposables;
5  using Platform.Singletons;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using Platform.Memory;
9  using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.United.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the united memory links base.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <seealso cref="DisposableBase"/>
23     /// <seealso cref="ILinks{TLink}"/>
24     public abstract class UnitedMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
25     {
26         private static readonly EqualityComparer<TLink> _equalityComparer =
27             EqualityComparer<TLink>.Default;
28         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
29         private static readonly uncheckedConverter<TLink, long> _addressToInt64Converter =
30             uncheckedConverter<TLink, long>.Default;
31         private static readonly uncheckedConverter<long, TLink> _int64ToAddressConverter =
32             uncheckedConverter<long, TLink>.Default;
33         private static readonly TLink _zero = default;
34         private static readonly TLink _one = Arithmetic.Increment(_zero);
35
36         /// <summary>Возвращает размер одной связи в байтах.</summary>
37         /// <remarks>
38         /// Используется только во вне класса, не рекомендуется использовать внутри.
39         /// Так как во вне не обязательно будет доступен unsafe C#.
40         /// </remarks>
41         public static readonly long LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
42
43         /// <summary>
44         /// <para>
45         /// The size in bytes.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
50
51         /// <summary>
52         /// <para>
53         /// The link size in bytes.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
58
59         /// <summary>
60         /// <para>
61         /// The memory.
62         /// </para>
63         /// <para></para>
64         /// </summary>
65         protected readonly IResizableDirectMemory _memory;
66
67         /// <summary>
68         /// <para>
69         /// The memory reservation step.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         protected readonly long _memoryReservationStep;
74
75         /// <summary>
76         /// <para>
77         /// The targets tree methods.
78         /// </para>
79         /// <para></para>
80         /// </summary>

```

```

77     protected ILinksTreeMethods<TLink> TargetsTreeMethods;
78     /// <summary>
79     /// <para>
80     /// The sources tree methods.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     protected ILinksTreeMethods<TLink> SourcesTreeMethods;
85     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
86     // ↳ нужно использовать не список а дерево, так как так можно быстрее проверить на
87     // ↳ наличие связи внутри
88     /// <summary>
89     /// <para>
90     /// The unused links list methods.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     protected ILinksListMethods<TLink> UnusedLinksListMethods;
95     /// <summary>
96     /// Возвращает общее число связей находящихся в хранилище.
97     /// </summary>
98     protected virtual TLink Total
99     {
100         [MethodImpl(MethodImplOptions.AggressiveInlining)]
101         get
102         {
103             ref var header = ref GetHeaderReference();
104             return Subtract(header.AllocatedLinks, header.FreeLinks);
105         }
106     }
107     /// <summary>
108     /// <para>
109     /// Gets the constants value.
110     /// </para>
111     /// <para></para>
112     /// </summary>
113     public virtual LinksConstants<TLink> Constants
114     {
115         [MethodImpl(MethodImplOptions.AggressiveInlining)]
116         get;
117     }
118     /// <summary>
119     /// <para>
120     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="memory">
125     /// <para>A memory.</para>
126     /// <para></para>
127     /// </param>
128     /// <param name="memoryReservationStep">
129     /// <para>A memory reservation step.</para>
130     /// <para></para>
131     /// </param>
132     /// <param name="constants">
133     /// <para>A constants.</para>
134     /// <para></para>
135     /// </param>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
138     ↳ memoryReservationStep, LinksConstants<TLink> constants)
139     {
140         _memory = memory;
141         _memoryReservationStep = memoryReservationStep;
142         Constants = constants;
143     }
144     /// <summary>
145     /// <para>
146     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
147     /// </para>
148     /// <para></para>
149     /// </summary>
150     /// <param name="memory">
151     /// <para>A memory.</para>
152     /// <para></para>

```

```

153     /// <para></para>
154     /// </param>
155     /// <param name="memoryReservationStep">
156     /// <para>A memory reservation step.</para>
157     /// <para></para>
158     /// </param>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
        ↪ memoryReservationStep) : this(memory, memoryReservationStep,
        ↪ Default<LinksConstants<TLink>>.Instance) { }

161
162     /// <summary>
163     /// <para>
164     /// Inits the memory.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="memory">
169     /// <para>The memory.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="memoryReservationStep">
173     /// <para>The memory reservation step.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
178     {
179         if (memory.ReservedCapacity < memoryReservationStep)
180         {
181             memory.ReservedCapacity = memoryReservationStep;
182         }
183         SetPointers(memory);
184         ref var header = ref GetHeaderReference();
185         // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
186         memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
        ↪ LinkHeaderSizeInBytes;
187         // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
188         header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
        ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
189     }
190
191     /// <summary>
192     /// <para>
193     /// Counts the substitution.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="restriction">
198     /// <para>The substitution.</para>
199     /// <para></para>
200     /// </param>
201     /// <exception cref="NotSupportedException">
202     /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
203     /// <para></para>
204     /// </exception>
205     /// <returns>
206     /// <para>The link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     public virtual TLink Count(ICollection<TLink>? restriction)
211     {
212         // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
213         if (restriction.Count == 0)
214         {
215             return Total;
216         }
217         var constants = Constants;
218         var any = constants.Any;
219         var index = restriction[constants.IndexPart];
220         if (restriction.Count == 1)
221         {
222             if (AreEqual(index, any))
223             {
224                 return Total;
225             }
226             return Exists(index) ? GetOne() : GetZero();

```



```

227 }
228 if (restriction.Count == 2)
229 {
230     var value = restriction[1];
231     if (AreEqual(index, any))
232     {
233         if (AreEqual(value, any))
234         {
235             return Total; // Any - как отсутствие ограничения
236         }
237         return Add(SourcesTreeMethods.CountUsages(value),
238             ↪ TargetsTreeMethods.CountUsages(value));
239     }
240     else
241     {
242         if (!Exists(index))
243         {
244             return GetZero();
245         }
246         if (AreEqual(value, any))
247         {
248             return GetOne();
249         }
250         ref var storedLinkValue = ref GetLinkReference(index);
251         if (AreEqual(storedLinkValue.Source, value) ||
252             ↪ AreEqual(storedLinkValue.Target, value))
253         {
254             return GetOne();
255         }
256         return GetZero();
257     }
258 }
259 if (restriction.Count == 3)
260 {
261     var source = restriction[constants.SourcePart];
262     var target = restriction[constants.TargetPart];
263     if (AreEqual(index, any))
264     {
265         if (AreEqual(source, any) && AreEqual(target, any))
266         {
267             return Total;
268         }
269         else if (AreEqual(source, any))
270         {
271             return TargetsTreeMethods.CountUsages(target);
272         }
273         else if (AreEqual(target, any))
274         {
275             return SourcesTreeMethods.CountUsages(source);
276         }
277         else //if(source != Any && target != Any)
278         {
279             // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
280             var link = SourcesTreeMethods.Search(source, target);
281             return AreEqual(link, constants.Null) ? GetZero() : GetOne();
282         }
283     }
284     else
285     {
286         if (!Exists(index))
287         {
288             return GetZero();
289         }
290         if (AreEqual(source, any) && AreEqual(target, any))
291         {
292             return GetOne();
293         }
294         ref var storedLinkValue = ref GetLinkReference(index);
295         if (!AreEqual(source, any) && !AreEqual(target, any))
296         {
297             if (AreEqual(storedLinkValue.Source, source) &&
298                 ↪ AreEqual(storedLinkValue.Target, target))
299             {
300                 return GetOne();
301             }
302             return GetZero();
303         }
304     }
305     var value = default(TLink);

```

```

302         if (AreEqual(source, any))
303         {
304             value = target;
305         }
306         if (AreEqual(target, any))
307         {
308             value = source;
309         }
310         if (AreEqual(storedLinkValue.Source, value) ||
311             ↪ AreEqual(storedLinkValue.Target, value))
312         {
313             return GetOne();
314         }
315         return GetZero();
316     }
317     throw new NotSupportedException("Другие размеры и способы ограничений не
318     ↪ поддерживаются.");
319 }
320 /// <summary>
321 /// <para>
322 /// Eaches the handler.
323 /// </para>
324 /// <para></para>
325 /// </summary>
326 /// <param name="handler">
327 /// <para>The handler.</para>
328 /// <para></para>
329 /// </param>
330 /// <param name="restriction">
331 /// <para>The substitution.</para>
332 /// <para></para>
333 /// </param>
334 /// <exception cref="NotSupportedException">
335 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
336 /// <para></para>
337 /// </exception>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public virtual TLink Each(ICollection<TLink> restriction, ReadHandler<TLink>? handler)
344 {
345     var constants = Constants;
346     var @break = constants.Break;
347     if (restriction.Count == 0)
348     {
349         for (var link = GetOne(); LessOrEqualThan(link,
350             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
351         {
352             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
353             {
354                 return @break;
355             }
356         }
357         return @break;
358     }
359     var @continue = constants.Continue;
360     var any = constants.Any;
361     var index = restriction[constants.IndexPart];
362     if (restriction.Count == 1)
363     {
364         if (AreEqual(index, any))
365         {
366             return Each(Array.Empty<TLink>(), handler);
367         }
368         if (!Exists(index))
369         {
370             return @continue;
371         }
372         return handler(GetLinkStruct(index));
373     }
374     if (restriction.Count == 2)
375     {
376         var value = restriction[1];
377         if (AreEqual(index, any))

```

```

377 {
378     if (AreEqual(value, any))
379     {
380         return Each(Array.Empty<TLink>(), handler);
381     }
382     if (AreEqual(Each(new Link<TLink>(index, value, any), handler), @break))
383     {
384         return @break;
385     }
386     return Each(new Link<TLink>(index, any, value), handler);
387 }
388 else
389 {
390     if (!Exists(index))
391     {
392         return @continue;
393     }
394     if (AreEqual(value, any))
395     {
396         return handler(GetLinkStruct(index));
397     }
398     ref var storedLinkValue = ref GetLinkReference(index);
399     if (AreEqual(storedLinkValue.Source, value) ||
400         AreEqual(storedLinkValue.Target, value))
401     {
402         return handler(GetLinkStruct(index));
403     }
404     return @continue;
405 }
406 }
407 if (restriction.Count == 3)
408 {
409     var source = restriction[constants.SourcePart];
410     var target = restriction[constants.TargetPart];
411     if (AreEqual(index, any))
412     {
413         if (AreEqual(source, any) && AreEqual(target, any))
414         {
415             return Each(Array.Empty<TLink>(), handler);
416         }
417         else if (AreEqual(source, any))
418         {
419             return TargetsTreeMethods.EachUsage(target, handler);
420         }
421         else if (AreEqual(target, any))
422         {
423             return SourcesTreeMethods.EachUsage(source, handler);
424         }
425         else //if(source != Any && target != Any)
426         {
427             var link = SourcesTreeMethods.Search(source, target);
428             return AreEqual(link, constants.Null) ? @continue :
429                 ↪ handler(GetLinkStruct(link));
430         }
431     }
432     else
433     {
434         if (!Exists(index))
435         {
436             return @continue;
437         }
438         if (AreEqual(source, any) && AreEqual(target, any))
439         {
440             return handler(GetLinkStruct(index));
441         }
442         ref var storedLinkValue = ref GetLinkReference(index);
443         if (!AreEqual(source, any) && !AreEqual(target, any))
444         {
445             if (AreEqual(storedLinkValue.Source, source) &&
446                 AreEqual(storedLinkValue.Target, target))
447             {
448                 return handler(GetLinkStruct(index));
449             }
450             return @continue;
451         }
452         var value = default(TLink);
453         if (AreEqual(source, any))
454         {

```

```

454         value = target;
455     }
456     if (AreEqual(target, any))
457     {
458         value = source;
459     }
460     if (AreEqual(storedLinkValue.Source, value) ||
461         AreEqual(storedLinkValue.Target, value))
462     {
463         return handler(GetLinkStruct(index));
464     }
465     return @continue;
466 }
467 }
468 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
469 }
470
471 /// <remarks>
472 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
    ↳ в другом месте (но не в менеджере памяти, а в логике Links)
473 /// </remarks>
474 [MethodImpl(MethodImplOptions.AggressiveInlining)]
475 public virtual TLink Update(ICollection<TLink> restriction, ICollection<TLink> substitution,
    ↳ WriteHandler<TLink> handler)
476 {
477     var constants = Constants;
478     var @null = constants.Null;
479     var linkIndex = restriction[constants.IndexPart];
480     var before = GetLinkStruct(linkIndex);
481     ref var link = ref GetLinkReference(linkIndex);
482     ref var header = ref GetHeaderReference();
483     ref var firstAsSource = ref header.RootAsSource;
484     ref var firstAsTarget = ref header.RootAsTarget;
485     // Будет корректно работать только в том случае, если пространство выделенной связи
    ↳ предварительно заполнено нулями
486     if (!AreEqual(link.Source, @null))
487     {
488         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
489     }
490     if (!AreEqual(link.Target, @null))
491     {
492         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
493     }
494     link.Source = substitution[constants.SourcePart];
495     link.Target = substitution[constants.TargetPart];
496     if (!AreEqual(link.Source, @null))
497     {
498         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
499     }
500     if (!AreEqual(link.Target, @null))
501     {
502         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
503     }
504     return handler != null ? handler(before, GetLinkStruct(linkIndex)) :
    ↳ Constants.Continue;
505 }
506
507 /// <remarks>
508 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
    ↳ пространство
509 /// </remarks>
510 [MethodImpl(MethodImplOptions.AggressiveInlining)]
511 public virtual TLink Create(ICollection<TLink> substitution, WriteHandler<TLink> handler)
512 {
513     ref var header = ref GetHeaderReference();
514     var freeLink = header.FirstFreeLink;
515     if (!AreEqual(freeLink, Constants.Null))
516     {
517         UnusedLinksListMethods.Detach(freeLink);
518     }
519     else
520     {
521         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
522         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
523         {
524             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
525         }
526     }
527 }

```

```

526         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
527         {
528             _memory.ReservedCapacity += _memory.ReservationStep;
529             SetPointers(_memory);
530             header = ref GetHeaderReference();
531             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
532                 ↳ LinkSizeInBytes);
533         }
534         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
535         _memory.UsedCapacity += LinkSizeInBytes;
536     }
537     return handler != null ? handler(null, new Link<TLink>(freeLink, Constants.Null,
538         ↳ Constants.Null)) : Constants.Continue;
539 }
540
541 /// <summary>
542 /// <para>
543 /// Deletes the substitution.
544 /// </para>
545 /// <para></para>
546 /// </summary>
547 /// <param name="restriction">
548 /// <para>The substitution.</para>
549 /// <para></para>
550 /// </param>
551 [MethodImpl(MethodImplOptions.AggressiveInlining)]
552 public virtual TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler)
553 {
554     ref var header = ref GetHeaderReference();
555     var link = restriction[Constants.IndexPart];
556     var before = GetLinkStruct(link);
557     if (LessThan(link, header.AllocatedLinks))
558     {
559         UnusedLinksListMethods.AttachAsFirst(link);
560         return handler != null ? handler(before, null) : Constants.Continue;
561     }
562     else if (AreEqual(link, header.AllocatedLinks))
563     {
564         header.AllocatedLinks = Decrement(header.AllocatedLinks);
565         _memory.UsedCapacity -= LinkSizeInBytes;
566         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
567         ↳ пока не дойдём до первой существующей связи
568         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
569         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
570             ↳ IsUnusedLink(header.AllocatedLinks))
571         {
572             UnusedLinksListMethods.Detach(header.AllocatedLinks);
573             header.AllocatedLinks = Decrement(header.AllocatedLinks);
574             _memory.UsedCapacity -= LinkSizeInBytes;
575         }
576         return handler != null ? handler(before, null) : Constants.Continue;
577     }
578     return Constants.Continue;
579 }
580
581 /// <summary>
582 /// <para>
583 /// Gets the link struct using the specified link index.
584 /// </para>
585 /// <para></para>
586 /// </summary>
587 /// <param name="linkIndex">
588 /// <para>The link index.</para>
589 /// <para></para>
590 /// </param>
591 /// <returns>
592 /// <para>A list of t link</para>
593 /// <para></para>
594 /// </returns>
595 [MethodImpl(MethodImplOptions.AggressiveInlining)]
596 public IList<TLink> GetLinkStruct(TLink linkIndex)
597 {
598     ref var link = ref GetLinkReference(linkIndex);
599     return new Link<TLink>(linkIndex, link.Source, link.Target);
600 }
601
602 /// <remarks>

```

```

599  /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
600  → адрес реально поменялся
601  ///
602  /// Указатель this.links может быть в том же месте,
603  /// так как 0-я связь не используется и имеет такой же размер как Header,
604  /// поэтому header размещается в том же месте, что и 0-я связь
605  /// </remarks>
606  [MethodImpl(MethodImplOptions.AggressiveInlining)]
607  protected abstract void SetPointers(IResizableDirectMemory memory);
608
609  /// <summary>
610  /// <para>
611  /// Resets the pointers.
612  /// </para>
613  /// </summary>
614  [MethodImpl(MethodImplOptions.AggressiveInlining)]
615  protected virtual void ResetPointers()
616  {
617      SourcesTreeMethods = null;
618      TargetsTreeMethods = null;
619      UnusedLinksListMethods = null;
620  }
621
622  /// <summary>
623  /// <para>
624  /// Gets the header reference.
625  /// </para>
626  /// </summary>
627  /// <returns>
628  /// <para>A ref links header of t link</para>
629  /// </returns>
630  [MethodImpl(MethodImplOptions.AggressiveInlining)]
631  protected abstract ref LinksHeader<TLink> GetHeaderReference();
632
633  /// <summary>
634  /// <para>
635  /// Gets the link reference using the specified link index.
636  /// </para>
637  /// </summary>
638  /// <param name="linkIndex">
639  /// <para>The link index.</para>
640  /// </param>
641  /// <returns>
642  /// <para>A ref raw link of t link</para>
643  /// </returns>
644  [MethodImpl(MethodImplOptions.AggressiveInlining)]
645  protected abstract ref RawLink<TLink> GetLinkReference(TLink linkIndex);
646
647  /// <summary>
648  /// <para>
649  /// Determines whether this instance exists.
650  /// </para>
651  /// </summary>
652  /// <param name="link">
653  /// <para>The link.</para>
654  /// </param>
655  /// <returns>
656  /// <para>The bool</para>
657  /// </returns>
658  [MethodImpl(MethodImplOptions.AggressiveInlining)]
659  protected virtual bool Exists(TLink link)
660  => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
661  && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
662  && !IsUnusedLink(link);
663
664  /// <summary>
665  /// <para>
666  /// Determines whether this instance is unused link.
667  /// </para>

```

```

676    /// <para></para>
677    /// </summary>
678    /// <param name="linkIndex">
679    /// <para>The link index.</para>
680    /// <para></para>
681    /// </param>
682    /// <returns>
683    /// <para>The bool</para>
684    /// <para></para>
685    /// </returns>
686    [MethodImpl(MethodImplOptions.AggressiveInlining)]
687    protected virtual bool IsUnusedLink(TLink linkIndex)
688    {
689        if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
690        ↪ is not needed
691        {
692            ref var link = ref GetLinkReference(linkIndex);
693            return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
694        }
695        else
696        {
697            return true;
698        }
699    }
700    /// <summary>
701    /// <para>
702    /// Gets the one.
703    /// </para>
704    /// <para></para>
705    /// </summary>
706    /// <returns>
707    /// <para>The link</para>
708    /// <para></para>
709    /// </returns>
710    [MethodImpl(MethodImplOptions.AggressiveInlining)]
711    protected virtual TLink GetOne() => _one;
712
713    /// <summary>
714    /// <para>
715    /// Gets the zero.
716    /// </para>
717    /// <para></para>
718    /// </summary>
719    /// <returns>
720    /// <para>The link</para>
721    /// <para></para>
722    /// </returns>
723    [MethodImpl(MethodImplOptions.AggressiveInlining)]
724    protected virtual TLink GetZero() => default;
725
726    /// <summary>
727    /// <para>
728    /// Determines whether this instance are equal.
729    /// </para>
730    /// <para></para>
731    /// </summary>
732    /// <param name="first">
733    /// <para>The first.</para>
734    /// <para></para>
735    /// </param>
736    /// <param name="second">
737    /// <para>The second.</para>
738    /// <para></para>
739    /// </param>
740    /// <returns>
741    /// <para>The bool</para>
742    /// <para></para>
743    /// </returns>
744    [MethodImpl(MethodImplOptions.AggressiveInlining)]
745    protected virtual bool AreEqual(TLink first, TLink second) =>
746    ↪ _equalityComparer.Equals(first, second);
747
748    /// <summary>
749    /// <para>
750    /// Determines whether this instance less than.
751    /// </para>
752    /// <para></para>

```

```

752     /// </summary>
753     /// <param name="first">
754     /// <para>The first.</para>
755     /// <para></para>
756     /// </param>
757     /// <param name="second">
758     /// <para>The second.</para>
759     /// <para></para>
760     /// </param>
761     /// <returns>
762     /// <para>The bool</para>
763     /// <para></para>
764     /// </returns>
765     [MethodImpl(MethodImplOptions.AggressiveInlining)]
766     protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
767         ↪ second) < 0;
768
769     /// <summary>
770     /// <para>
771     /// Determines whether this instance less or equal than.
772     /// </para>
773     /// <para></para>
774     /// </summary>
775     /// <param name="first">
776     /// <para>The first.</para>
777     /// <para></para>
778     /// </param>
779     /// <param name="second">
780     /// <para>The second.</para>
781     /// <para></para>
782     /// </param>
783     /// <returns>
784     /// <para>The bool</para>
785     /// <para></para>
786     /// </returns>
787     [MethodImpl(MethodImplOptions.AggressiveInlining)]
788     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
789         ↪ _comparer.Compare(first, second) <= 0;
790
791     /// <summary>
792     /// <para>
793     /// Determines whether this instance greater than.
794     /// </para>
795     /// <para></para>
796     /// </summary>
797     /// <param name="first">
798     /// <para>The first.</para>
799     /// <para></para>
800     /// </param>
801     /// <param name="second">
802     /// <para>The second.</para>
803     /// <para></para>
804     /// </param>
805     /// <returns>
806     /// <para>The bool</para>
807     /// <para></para>
808     /// </returns>
809     [MethodImpl(MethodImplOptions.AggressiveInlining)]
810     protected virtual bool GreaterThan(TLink first, TLink second) =>
811         ↪ _comparer.Compare(first, second) > 0;
812
813     /// <summary>
814     /// <para>
815     /// Determines whether this instance greater or equal than.
816     /// </para>
817     /// <para></para>
818     /// </summary>
819     /// <param name="first">
820     /// <para>The first.</para>
821     /// <para></para>
822     /// </param>
823     /// <param name="second">
824     /// <para>The second.</para>
825     /// <para></para>
826     /// </param>
827     /// <returns>
828     /// <para>The bool</para>
829     /// <para></para>
830     /// </returns>

```



```

827     /// </returns>
828     [MethodImpl(MethodImplOptions.AggressiveInlining)]
829     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
830         ↪ _comparer.Compare(first, second) >= 0;
831
832     /// <summary>
833     /// <para>
834     /// Converts the to int 64 using the specified value.
835     /// </para>
836     /// <para></para>
837     /// </summary>
838     /// <param name="value">
839     /// <para>The value.</para>
840     /// <para></para>
841     /// </param>
842     /// <returns>
843     /// <para>The long</para>
844     /// <para></para>
845     /// </returns>
846     [MethodImpl(MethodImplOptions.AggressiveInlining)]
847     protected virtual long ConvertToInt64(TLink value) =>
848         ↪ _addressToInt64Converter.Convert(value);
849
850     /// <summary>
851     /// <para>
852     /// Converts the to address using the specified value.
853     /// </para>
854     /// <para></para>
855     /// </summary>
856     /// <param name="value">
857     /// <para>The value.</para>
858     /// <para></para>
859     /// </param>
860     /// <returns>
861     /// <para>The link</para>
862     /// <para></para>
863     /// </returns>
864     [MethodImpl(MethodImplOptions.AggressiveInlining)]
865     protected virtual TLink ConvertToAddress(long value) =>
866         ↪ _int64ToAddressConverter.Convert(value);
867
868     /// <summary>
869     /// <para>
870     /// Adds the first.
871     /// </para>
872     /// <para></para>
873     /// </summary>
874     /// <param name="first">
875     /// <para>The first.</para>
876     /// <para></para>
877     /// </param>
878     /// <param name="second">
879     /// <para>The second.</para>
880     /// <para></para>
881     /// </param>
882     /// <returns>
883     /// <para>The link</para>
884     /// <para></para>
885     /// </returns>
886     [MethodImpl(MethodImplOptions.AggressiveInlining)]
887     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
888         ↪ second);
889
890     /// <summary>
891     /// <para>
892     /// Subtracts the first.
893     /// </para>
894     /// <para></para>
895     /// </summary>
896     /// <param name="first">
897     /// <para>The first.</para>
898     /// <para></para>
899     /// </param>
900     /// <param name="second">
901     /// <para>The second.</para>
902     /// <para></para>
903     /// </param>
904     /// <returns>

```

```

901    /// <para>The link</para>
902    /// <para></para>
903    /// </returns>
904    [MethodImpl(MethodImplOptions.AggressiveInlining)]
905    protected virtual TLink Subtract(TLink first, TLink second) =>
906        ↪ Arithmetic<TLink>.Subtract(first, second);
907
908    /// <summary>
909    /// <para>
910    /// Increments the link.
911    /// </para>
912    /// <para></para>
913    /// </summary>
914    /// <param name="link">
915    /// <para>The link.</para>
916    /// <para></para>
917    /// </param>
918    /// <returns>
919    /// <para>The link</para>
920    /// <para></para>
921    /// </returns>
922    [MethodImpl(MethodImplOptions.AggressiveInlining)]
923    protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
924
925    /// <summary>
926    /// <para>
927    /// Decrements the link.
928    /// </para>
929    /// <para></para>
930    /// </summary>
931    /// <param name="link">
932    /// <para>The link.</para>
933    /// <para></para>
934    /// </param>
935    /// <returns>
936    /// <para>The link</para>
937    /// <para></para>
938    /// </returns>
939    [MethodImpl(MethodImplOptions.AggressiveInlining)]
940    protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
941
942    #region Disposable
943
944    /// <summary>
945    /// <para>
946    /// Gets the allow multiple dispose calls value.
947    /// </para>
948    /// <para></para>
949    /// </summary>
950    protected override bool AllowMultipleDisposeCalls
951    {
952        [MethodImpl(MethodImplOptions.AggressiveInlining)]
953        get => true;
954    }
955
956    /// <summary>
957    /// <para>
958    /// Disposes the manual.
959    /// </para>
960    /// <para></para>
961    /// </summary>
962    /// <param name="manual">
963    /// <para>The manual.</para>
964    /// <para></para>
965    /// </param>
966    /// <param name="wasDisposed">
967    /// <para>The was disposed.</para>
968    /// <para></para>
969    /// </param>
970    [MethodImpl(MethodImplOptions.AggressiveInlining)]
971    protected override void Dispose(bool manual, bool wasDisposed)
972    {
973        if (!wasDisposed)
974        {
975            ResetPointers();
976            _memory.DisposeIfPossible();
977        }
978    }

```

```

978         #endregion
979     }
980 }
981 }

```

1.91 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United.Generic
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unused links list methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLink}"/>
17     /// <seealso cref="ILinksListMethods{TLink}"/>
18     public unsafe class UnusedLinksListMethods<TLink> :
19         ↳ AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
20     {
21         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
22             ↳ UncheckedConverter<TLink, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public UnusedLinksListMethods(byte* links, byte* header)
40         {
41             _links = links;
42             _header = header;
43         }
44
45         /// <summary>
46         /// <para>
47         /// Gets the header reference.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <returns>
52         /// <para>A ref links header of t link</para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
56             ↳ AsRef<LinksHeader<TLink>>(_header);
57
58         /// <summary>
59         /// <para>
60         /// Gets the link reference using the specified link.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         /// <param name="link">
65         /// <para>The link.</para>
66         /// </param>
67         /// <returns>
68         /// <para>A ref raw link of t link</para>
69         /// </returns>
70     }

```

```

71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
    ↪ AsRef<RawLink<TLink>>(_links + (RawLink<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));

74
75     /// <summary>
76     /// <para>
77     /// Gets the first.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
87
88     /// <summary>
89     /// <para>
90     /// Gets the last.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <returns>
95     /// <para>The link</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
100
101     /// <summary>
102     /// <para>
103     /// Gets the previous using the specified element.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="element">
108     /// <para>The element.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>The link</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected override TLink GetPrevious(TLink element) => GetLinkReference(element).Source;
117
118     /// <summary>
119     /// <para>
120     /// Gets the next using the specified element.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="element">
125     /// <para>The element.</para>
126     /// <para></para>
127     /// </param>
128     /// <returns>
129     /// <para>The link</para>
130     /// <para></para>
131     /// </returns>
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     protected override TLink GetNext(TLink element) => GetLinkReference(element).Target;
134
135     /// <summary>
136     /// <para>
137     /// Gets the size.
138     /// </para>
139     /// <para></para>
140     /// </summary>
141     /// <returns>
142     /// <para>The link</para>
143     /// <para></para>
144     /// </returns>
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     protected override TLink GetSize() => GetHeaderReference().FreeLinks;

```

```

147     /// <summary>
148     /// <para>
149     /// Sets the first using the specified element.
150     /// </para>
151     /// <para></para>
152     /// </summary>
153     /// <param name="element">
154     /// <para>The element.</para>
155     /// <para></para>
156     /// </param>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
159         element;
160
161     /// <summary>
162     /// <para>
163     /// Sets the last using the specified element.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="element">
168     /// <para>The element.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
173         element;
174
175     /// <summary>
176     /// <para>
177     /// Sets the previous using the specified element.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <param name="element">
182     /// <para>The element.</para>
183     /// <para></para>
184     /// </param>
185     /// <param name="previous">
186     /// <para>The previous.</para>
187     /// <para></para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override void SetPrevious(TLink element, TLink previous) =>
191         GetLinkReference(element).Source = previous;
192
193     /// <summary>
194     /// <para>
195     /// Sets the next using the specified element.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="element">
200     /// <para>The element.</para>
201     /// <para></para>
202     /// </param>
203     /// <param name="next">
204     /// <para>The next.</para>
205     /// <para></para>
206     /// </param>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override void SetNext(TLink element, TLink next) =>
209         GetLinkReference(element).Target = next;
210
211     /// <summary>
212     /// <para>
213     /// Sets the size using the specified size.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="size">
218     /// <para>The size.</para>
219     /// <para></para>
220     /// </param>
221     [MethodImpl(MethodImplOptions.AggressiveInlining)]
222     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;

```

1.92 ./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLink<TLink> : IEquatable<RawLink<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink Source;
36
37         /// <summary>
38         /// <para>
39         /// The target.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink Target;
44
45         /// <summary>
46         /// <para>
47         /// The left as source.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLink LeftAsSource;
52
53         /// <summary>
54         /// <para>
55         /// The right as source.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         public TLink RightAsSource;
60
61         /// <summary>
62         /// <para>
63         /// The size as source.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         public TLink SizeAsSource;
68
69         /// <summary>
70         /// <para>
71         /// The left as target.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         public TLink LeftAsTarget;
76
77         /// <summary>
78         /// <para>
79         /// The right as target.
80         /// </para>
81         /// <para></para>
82         /// </summary>
83         public TLink RightAsTarget;
84     }
85 }

```

```

76     public TLink RightAsTarget;
77     /// <summary>
78     /// <para>
79     /// The size as target.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLink SizeAsTarget;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is RawLink<TLink> link ? Equals(link) :
        ↳ false;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance equals.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="other">
109    /// <para>The other.</para>
110    /// <para></para>
111    /// </param>
112    /// <returns>
113    /// <para>The bool</para>
114    /// <para></para>
115    /// </returns>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    public bool Equals(RawLink<TLink> other)
118        => _equalityComparer.Equals(Source, other.Source)
119            && _equalityComparer.Equals(Target, other.Target)
120            && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
121            && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
122            && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
123            && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
124            && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
125            && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
126
127    /// <summary>
128    /// <para>
129    /// Gets the hash code.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <returns>
134    /// <para>The int</para>
135    /// <para></para>
136    /// </returns>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
        ↳ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
139
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public static bool operator ==(RawLink<TLink> left, RawLink<TLink> right) =>
        ↳ left.Equals(right);
142
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator !=(RawLink<TLink> left, RawLink<TLink> right) => !(left ==
        ↳ right);
145 }
146 }

```

1.93 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethod

```
1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 links recursionless size balanced tree methods base.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{uint}"/>
15    public unsafe abstract class UInt32LinksRecursionlessSizeBalancedTreeMethodsBase :
16    ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<uint>
17    {
18        /// <summary>
19        /// <para>
20        /// The links.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        protected new readonly RawLink<uint>* Links;
25        /// <summary>
26        /// <para>
27        /// The header.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        protected new readonly LinksHeader<uint>* Header;
32
33        /// <summary>
34        /// <para>
35        /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
36        ↪ instance.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        /// <param name="constants">
41        /// <para>A constants.</para>
42        /// <para></para>
43        /// </param>
44        /// <param name="links">
45        /// <para>A links.</para>
46        /// <para></para>
47        /// </param>
48        /// <param name="header">
49        /// <para>A header.</para>
50        /// <para></para>
51        /// </param>
52        protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
53        ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
54        : base(constants, (byte*)links, (byte*)header)
55        {
56            Links = links;
57            Header = header;
58        }
59
60        /// <summary>
61        /// <para>
62        /// Gets the zero.
63        /// </para>
64        /// <para></para>
65        /// </summary>
66        /// <returns>
67        /// <para>The uint</para>
68        /// <para></para>
69        /// </returns>
70        [MethodImpl(MethodImplOptions.AggressiveInlining)]
71        protected override uint GetZero() => 0U;
72
73        /// <summary>
74        /// <para>
75        /// Determines whether this instance equal to zero.
76        /// </para>
77        /// <para></para>
```



```

75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(uint value) => value == 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance are equal.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="first">
94     /// <para>The first.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="second">
98     /// <para>The second.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override bool AreEqual(uint first, uint second) => first == second;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance greater than zero.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="value">
115    /// <para>The value.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>The bool</para>
120    /// <para></para>
121    /// </returns>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    protected override bool GreaterThanZero(uint value) => value > 0U;
124
125    /// <summary>
126    /// <para>
127    /// Determines whether this instance greater than.
128    /// </para>
129    /// <para></para>
130    /// </summary>
131    /// <param name="first">
132    /// <para>The first.</para>
133    /// <para></para>
134    /// </param>
135    /// <param name="second">
136    /// <para>The second.</para>
137    /// <para></para>
138    /// </param>
139    /// <returns>
140    /// <para>The bool</para>
141    /// <para></para>
142    /// </returns>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override bool GreaterThan(uint first, uint second) => first > second;
145
146    /// <summary>
147    /// <para>
148    /// Determines whether this instance greater or equal than.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="first">

```

```

153     /// <para>The first.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="second">
157     /// <para>The second.</para>
158     /// <para></para>
159     /// </param>
160     /// <returns>
161     /// <para>The bool</para>
162     /// <para></para>
163     /// </returns>
164     [MethodImpl(MethodImplOptions.AggressiveInlining)]
165     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
166
167     /// <summary>
168     /// <para>
169     /// Determines whether this instance greater or equal than zero.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="value">
174     /// <para>The value.</para>
175     /// <para></para>
176     /// </param>
177     /// <returns>
178     /// <para>The bool</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
183     ↪ always true for uint
184
185     /// <summary>
186     /// <para>
187     /// Determines whether this instance less or equal than zero.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="value">
192     /// <para>The value.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The bool</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override bool LessOrEqualThanZero(uint value) => value == 0U; // value is
201     ↪ always >= 0 for uint
202
203     /// <summary>
204     /// <para>
205     /// Determines whether this instance less or equal than.
206     /// </para>
207     /// <para></para>
208     /// </summary>
209     /// <param name="first">
210     /// <para>The first.</para>
211     /// <para></para>
212     /// </param>
213     /// <param name="second">
214     /// <para>The second.</para>
215     /// <para></para>
216     /// </param>
217     /// <returns>
218     /// <para>The bool</para>
219     /// <para></para>
220     /// </returns>
221     [MethodImpl(MethodImplOptions.AggressiveInlining)]
222     protected override bool LessOrEqualThan(uint first, uint second) => first <= second;
223
224     /// <summary>
225     /// <para>
226     /// Determines whether this instance less than zero.
227     /// </para>
228     /// <para></para>
229     /// </summary>
230     /// <param name="value">

```

```

229     /// <para>The value.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool LessThanZero(uint value) => false; // value < 0 is always false
238     ↪ for uint
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance less than.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="first">
247     /// <para>The first.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="second">
251     /// <para>The second.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool LessThan(uint first, uint second) => first < second;
260
261     /// <summary>
262     /// <para>
263     /// Increments the value.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="value">
268     /// <para>The value.</para>
269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The uint</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override uint Increment(uint value) => ++value;
277
278     /// <summary>
279     /// <para>
280     /// Decrements the value.
281     /// </para>
282     /// <para></para>
283     /// </summary>
284     /// <param name="value">
285     /// <para>The value.</para>
286     /// <para></para>
287     /// </param>
288     /// <returns>
289     /// <para>The uint</para>
290     /// <para></para>
291     /// </returns>
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     protected override uint Decrement(uint value) => --value;
294
295     /// <summary>
296     /// <para>
297     /// Adds the first.
298     /// </para>
299     /// <para></para>
300     /// </summary>
301     /// <param name="first">
302     /// <para>The first.</para>
303     /// <para></para>
304     /// </param>
305     /// <param name="second">
306     /// <para>The second.</para>

```

```

306    /// <para></para>
307    /// </param>
308    /// <returns>
309    /// <para>The uint</para>
310    /// <para></para>
311    /// </returns>
312    [MethodImpl(MethodImplOptions.AggressiveInlining)]
313    protected override uint Add(uint first, uint second) => first + second;
314
315    /// <summary>
316    /// <para>
317    /// Subtracts the first.
318    /// </para>
319    /// <para></para>
320    /// </summary>
321    /// <param name="first">
322    /// <para>The first.</para>
323    /// <para></para>
324    /// </param>
325    /// <param name="second">
326    /// <para>The second.</para>
327    /// <para></para>
328    /// </param>
329    /// <returns>
330    /// <para>The uint</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override uint Subtract(uint first, uint second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>
348    /// <para></para>
349    /// </param>
350    /// <returns>
351    /// <para>The bool</para>
352    /// <para></para>
353    /// </returns>
354    [MethodImpl(MethodImplOptions.AggressiveInlining)]
355    protected override bool FirstIsToTheLeftOfSecond(uint first, uint second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }
362
363    /// <summary>
364    /// <para>
365    /// Determines whether this instance first is to the right of second.
366    /// </para>
367    /// <para></para>
368    /// </summary>
369    /// <param name="first">
370    /// <para>The first.</para>
371    /// <para></para>
372    /// </param>
373    /// <param name="second">
374    /// <para>The second.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override bool FirstIsToTheRightOfSecond(uint first, uint second)
383    {

```

```

383         ref var firstLink = ref Links[first];
384         ref var secondLink = ref Links[second];
385         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
386     }
387
388     /// <summary>
389     /// <para>
390     /// Gets the header reference.
391     /// </para>
392     /// <para></para>
393     /// </summary>
394     /// <returns>
395     /// <para>A ref links header of uint</para>
396     /// <para></para>
397     /// </returns>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
400
401     /// <summary>
402     /// <para>
403     /// Gets the link reference using the specified link.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     /// <param name="link">
408     /// <para>The link.</para>
409     /// <para></para>
410     /// </param>
411     /// <returns>
412     /// <para>A ref raw link of uint</para>
413     /// <para></para>
414     /// </returns>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<uint> GetLinkReference(uint link) => ref Links[link];
417 }
418 }

```

1.94 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 32 links size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{uint}"/>
15     public unsafe abstract class UInt32LinksSizeBalancedTreeMethodsBase :
        ↪ LinksSizeBalancedTreeMethodsBase<uint>
16     {
17         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         protected new readonly RawLink<uint>* Links;
24         /// <summary>
25         /// <para>
26         /// The header.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         protected new readonly LinksHeader<uint>* Header;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="constants">
39         /// <para>A constants.</para>

```

```

40     /// <para></para>
41     /// </param>
42     /// <param name="links">
43     /// <para>A links.</para>
44     /// <para></para>
45     /// </param>
46     /// <param name="header">
47     /// <para>A header.</para>
48     /// <para></para>
49     /// </param>
50     protected UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
51     ↪ RawLink<uint>* links, LinksHeader<uint>* header)
52     : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55         Header = header;
56     }
57     /// <summary>
58     /// <para>
59     /// Gets the zero.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The uint</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override uint GetZero() => 0U;
69
70     /// <summary>
71     /// <para>
72     /// Determines whether this instance equal to zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(uint value) => value == 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance are equal.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="first">
94     /// <para>The first.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="second">
98     /// <para>The second.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override bool AreEqual(uint first, uint second) => first == second;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance greater than zero.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="value">
115    /// <para>The value.</para>
116    /// <para></para>

```

```

117     /// </param>
118     /// <returns>
119     /// <para>The bool</para>
120     /// <para></para>
121     /// </returns>
122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
123     protected override bool GreaterThanZero(uint value) => value > 0U;
124
125     /// <summary>
126     /// <para>
127     /// Determines whether this instance greater than.
128     /// </para>
129     /// <para></para>
130     /// </summary>
131     /// <param name="first">
132     /// <para>The first.</para>
133     /// <para></para>
134     /// </param>
135     /// <param name="second">
136     /// <para>The second.</para>
137     /// <para></para>
138     /// </param>
139     /// <returns>
140     /// <para>The bool</para>
141     /// <para></para>
142     /// </returns>
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     protected override bool GreaterThan(uint first, uint second) => first > second;
145
146     /// <summary>
147     /// <para>
148     /// Determines whether this instance greater or equal than.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="first">
153     /// <para>The first.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="second">
157     /// <para>The second.</para>
158     /// <para></para>
159     /// </param>
160     /// <returns>
161     /// <para>The bool</para>
162     /// <para></para>
163     /// </returns>
164     [MethodImpl(MethodImplOptions.AggressiveInlining)]
165     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
166
167     /// <summary>
168     /// <para>
169     /// Determines whether this instance greater or equal than zero.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="value">
174     /// <para>The value.</para>
175     /// <para></para>
176     /// </param>
177     /// <returns>
178     /// <para>The bool</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
183     ↪ always true for uint
184
185     /// <summary>
186     /// <para>
187     /// Determines whether this instance less or equal than zero.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="value">
192     /// <para>The value.</para>
193     /// <para></para>
194     /// </param>

```

```

194    /// <returns>
195    /// <para>The bool</para>
196    /// <para></para>
197    /// </returns>
198    [MethodImpl(MethodImplOptions.AggressiveInlining)]
199    protected override bool LessOrEqualThanZero(uint value) => value == 0U; // value is
    ↳ always >= 0 for uint

200
201    /// <summary>
202    /// <para>
203    /// Determines whether this instance less or equal than.
204    /// </para>
205    /// <para></para>
206    /// </summary>
207    /// <param name="first">
208    /// <para>The first.</para>
209    /// <para></para>
210    /// </param>
211    /// <param name="second">
212    /// <para>The second.</para>
213    /// <para></para>
214    /// </param>
215    /// <returns>
216    /// <para>The bool</para>
217    /// <para></para>
218    /// </returns>
219    [MethodImpl(MethodImplOptions.AggressiveInlining)]
220    protected override bool LessOrEqualThan(uint first, uint second) => first <= second;
221
222    /// <summary>
223    /// <para>
224    /// Determines whether this instance less than zero.
225    /// </para>
226    /// <para></para>
227    /// </summary>
228    /// <param name="value">
229    /// <para>The value.</para>
230    /// <para></para>
231    /// </param>
232    /// <returns>
233    /// <para>The bool</para>
234    /// <para></para>
235    /// </returns>
236    [MethodImpl(MethodImplOptions.AggressiveInlining)]
237    protected override bool LessThanZero(uint value) => false; // value < 0 is always false
    ↳ for uint

238
239    /// <summary>
240    /// <para>
241    /// Determines whether this instance less than.
242    /// </para>
243    /// <para></para>
244    /// </summary>
245    /// <param name="first">
246    /// <para>The first.</para>
247    /// <para></para>
248    /// </param>
249    /// <param name="second">
250    /// <para>The second.</para>
251    /// <para></para>
252    /// </param>
253    /// <returns>
254    /// <para>The bool</para>
255    /// <para></para>
256    /// </returns>
257    [MethodImpl(MethodImplOptions.AggressiveInlining)]
258    protected override bool LessThan(uint first, uint second) => first < second;
259
260    /// <summary>
261    /// <para>
262    /// Increments the value.
263    /// </para>
264    /// <para></para>
265    /// </summary>
266    /// <param name="value">
267    /// <para>The value.</para>
268    /// <para></para>
269    /// </param>

```



```

270    /// <returns>
271    /// <para>The uint</para>
272    /// <para></para>
273    /// </returns>
274    [MethodImpl(MethodImplOptions.AggressiveInlining)]
275    protected override uint Increment(uint value) => ++value;
276
277    /// <summary>
278    /// <para>
279    /// Decrements the value.
280    /// </para>
281    /// <para></para>
282    /// </summary>
283    /// <param name="value">
284    /// <para>The value.</para>
285    /// <para></para>
286    /// </param>
287    /// <returns>
288    /// <para>The uint</para>
289    /// <para></para>
290    /// </returns>
291    [MethodImpl(MethodImplOptions.AggressiveInlining)]
292    protected override uint Decrement(uint value) => --value;
293
294    /// <summary>
295    /// <para>
296    /// Adds the first.
297    /// </para>
298    /// <para></para>
299    /// </summary>
300    /// <param name="first">
301    /// <para>The first.</para>
302    /// <para></para>
303    /// </param>
304    /// <param name="second">
305    /// <para>The second.</para>
306    /// <para></para>
307    /// </param>
308    /// <returns>
309    /// <para>The uint</para>
310    /// <para></para>
311    /// </returns>
312    [MethodImpl(MethodImplOptions.AggressiveInlining)]
313    protected override uint Add(uint first, uint second) => first + second;
314
315    /// <summary>
316    /// <para>
317    /// Subtracts the first.
318    /// </para>
319    /// <para></para>
320    /// </summary>
321    /// <param name="first">
322    /// <para>The first.</para>
323    /// <para></para>
324    /// </param>
325    /// <param name="second">
326    /// <para>The second.</para>
327    /// <para></para>
328    /// </param>
329    /// <returns>
330    /// <para>The uint</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override uint Subtract(uint first, uint second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>

```

```

348     /// <para></para>
349     /// </param>
350     /// <returns>
351     /// <para>The bool</para>
352     /// <para></para>
353     /// </returns>
354     [MethodImpl(MethodImplOptions.AggressiveInlining)]
355     protected override bool FirstIsToLeftOfSecond(uint first, uint second)
356     {
357         ref var firstLink = ref Links[first];
358         ref var secondLink = ref Links[second];
359         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360             ↪ secondLink.Source, secondLink.Target);
361     }
362     /// <summary>
363     /// <para>
364     /// Determines whether this instance first is to the right of second.
365     /// </para>
366     /// <para></para>
367     /// </summary>
368     /// <param name="first">
369     /// <para>The first.</para>
370     /// <para></para>
371     /// </param>
372     /// <param name="second">
373     /// <para>The second.</para>
374     /// <para></para>
375     /// </param>
376     /// <returns>
377     /// <para>The bool</para>
378     /// <para></para>
379     /// </returns>
380     [MethodImpl(MethodImplOptions.AggressiveInlining)]
381     protected override bool FirstIsToTheRightOfSecond(uint first, uint second)
382     {
383         ref var firstLink = ref Links[first];
384         ref var secondLink = ref Links[second];
385         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
386             ↪ secondLink.Source, secondLink.Target);
387     }
388     /// <summary>
389     /// <para>
390     /// Gets the header reference.
391     /// </para>
392     /// <para></para>
393     /// </summary>
394     /// <returns>
395     /// <para>A ref links header of uint</para>
396     /// <para></para>
397     /// </returns>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
400
401     /// <summary>
402     /// <para>
403     /// Gets the link reference using the specified link.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     /// <param name="link">
408     /// <para>The link.</para>
409     /// <para></para>
410     /// </param>
411     /// <returns>
412     /// <para>A ref raw link of uint</para>
413     /// <para></para>
414     /// </returns>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<uint> GetLinkReference(uint link) => ref Links[link];
417 }
418 }

```

```

1.95 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTree
1     using System.Runtime.CompilerServices;
2
3     #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

```

```

4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods :
15        ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>
36        public UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
37            ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
38            ↪ header) { }
39
40        /// <summary>
41        /// <para>
42        /// Gets the left reference using the specified node.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="node">
47        /// <para>The node.</para>
48        /// <para></para>
49        /// </param>
50        /// <returns>
51        /// <para>The ref uint</para>
52        /// <para></para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
56
57        /// <summary>
58        /// <para>
59        /// Gets the right reference using the specified node.
60        /// </para>
61        /// <para></para>
62        /// </summary>
63        /// <param name="node">
64        /// <para>The node.</para>
65        /// <para></para>
66        /// </param>
67        /// <returns>
68        /// <para>The ref uint</para>
69        /// <para></para>
70        /// </returns>
71        [MethodImpl(MethodImplOptions.AggressiveInlining)]
72        protected override ref uint GetRightReference(uint node) => ref
73            ↪ Links[node].RightAsSource;
74
75        /// <summary>
76        /// <para>
77        /// Gets the left using the specified node.
78        /// </para>
79        /// <para></para>
80        /// </summary>
81        /// <param name="node">

```

```

77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The uint</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override uint GetLeft(uint node) => Links[node].LeftAsSource;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The uint</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override uint GetRight(uint node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(uint node, uint left) => Links[node].LeftAsSource = left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(uint node, uint right) => Links[node].RightAsSource =
    ↪ right;
137
138    /// <summary>
139    /// <para>
140    /// Gets the size using the specified node.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <param name="node">
145    /// <para>The node.</para>
146    /// <para></para>
147    /// </param>
148    /// <returns>
149    /// <para>The uint</para>
150    /// <para></para>
151    /// </returns>
152    [MethodImpl(MethodImplOptions.AggressiveInlining)]
153    protected override uint GetSize(uint node) => Links[node].SizeAsSource;

```

```

154     /// <summary>
155     /// <para>
156     /// Sets the size using the specified node.
157     /// </para>
158     /// <para></para>
159     /// </summary>
160     /// <param name="node">
161     /// <para>The node.</para>
162     /// <para></para>
163     /// </param>
164     /// <param name="size">
165     /// <para>The size.</para>
166     /// <para></para>
167     /// </param>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     protected override void SetSize(uint node, uint size) => Links[node].SizeAsSource = size;
170
171     /// <summary>
172     /// <para>
173     /// Gets the tree root.
174     /// </para>
175     /// <para></para>
176     /// </summary>
177     /// <returns>
178     /// <para>The uint</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override uint GetTreeRoot() => Header->RootAsSource;
183
184     /// <summary>
185     /// <para>
186     /// Gets the base part value using the specified link.
187     /// </para>
188     /// <para></para>
189     /// </summary>
190     /// <param name="link">
191     /// <para>The link.</para>
192     /// <para></para>
193     /// </param>
194     /// <returns>
195     /// <para>The uint</para>
196     /// <para></para>
197     /// </returns>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override uint GetBasePartValue(uint link) => Links[link].Source;
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance first is to the left of second.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="firstSource">
208     /// <para>The first source.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="firstTarget">
212     /// <para>The first target.</para>
213     /// <para></para>
214     /// </param>
215     /// <param name="secondSource">
216     /// <para>The second source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="secondTarget">
220     /// <para>The second target.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The bool</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override bool FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
229     ↪ uint secondSource, uint secondTarget)

```

```

230         => firstSource < secondSource || (firstSource == secondSource && firstTarget <
           ↪ secondTarget);
231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
           ↪ uint secondSource, uint secondTarget)
260         => firstSource > secondSource || (firstSource == secondSource && firstTarget >
           ↪ secondTarget);
261
262     /// <summary>
263     /// <para>
264     /// Clears the node using the specified node.
265     /// </para>
266     /// <para></para>
267     /// </summary>
268     /// <param name="node">
269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(uint node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsSource = 0U;
277         link.RightAsSource = 0U;
278         link.SizeAsSource = 0U;
279     }
280 }
281 }

```

1.96 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links sources size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksSourcesSizeBalancedTreeMethods :
           ↪ UInt32LinksSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>

```

```

21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
35         ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
36
37     /// <summary>
38     /// <para>
39     /// Gets the left reference using the specified node.
40     /// </para>
41     /// <para></para>
42     /// </summary>
43     /// <param name="node">
44     /// <para>The node.</para>
45     /// <para></para>
46     /// </param>
47     /// <returns>
48     /// <para>The ref uint</para>
49     /// <para></para>
50     /// </returns>
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
53
54     /// <summary>
55     /// <para>
56     /// Gets the right reference using the specified node.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     /// <param name="node">
61     /// <para>The node.</para>
62     /// <para></para>
63     /// </param>
64     /// <returns>
65     /// <para>The ref uint</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref uint GetRightReference(uint node) => ref
70         ↳ Links[node].RightAsSource;
71
72     /// <summary>
73     /// <para>
74     /// Gets the left using the specified node.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="node">
79     /// <para>The node.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The uint</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override uint GetLeft(uint node) => Links[node].LeftAsSource;
88
89     /// <summary>
90     /// <para>
91     /// Gets the right using the specified node.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="node">
96     /// <para>The node.</para>
97     /// <para></para>
98     /// </param>

```

```

97     /// <returns>
98     /// <para>The uint</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override uint GetRight(uint node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(uint node, uint left) => Links[node].LeftAsSource = left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(uint node, uint right) => Links[node].RightAsSource =
    ↪ right;
137
138    /// <summary>
139    /// <para>
140    /// Gets the size using the specified node.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <param name="node">
145    /// <para>The node.</para>
146    /// <para></para>
147    /// </param>
148    /// <returns>
149    /// <para>The uint</para>
150    /// <para></para>
151    /// </returns>
152    [MethodImpl(MethodImplOptions.AggressiveInlining)]
153    protected override uint GetSize(uint node) => Links[node].SizeAsSource;
154
155    /// <summary>
156    /// <para>
157    /// Sets the size using the specified node.
158    /// </para>
159    /// <para></para>
160    /// </summary>
161    /// <param name="node">
162    /// <para>The node.</para>
163    /// <para></para>
164    /// </param>
165    /// <param name="size">
166    /// <para>The size.</para>
167    /// <para></para>
168    /// </param>
169    [MethodImpl(MethodImplOptions.AggressiveInlining)]
170    protected override void SetSize(uint node, uint size) => Links[node].SizeAsSource = size;
171
172    /// <summary>
173    /// <para>

```



```

174     /// Gets the tree root.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <returns>
179     /// <para>The uint</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override uint GetTreeRoot() => Header->RootAsSource;
184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified link.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The uint</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override uint GetBasePartValue(uint link) => Links[link].Source;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="firstSource">
209     /// <para>The first source.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="firstTarget">
213     /// <para>The first target.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToLeftOfSecond(uint firstSource, uint firstTarget,
230     ↪ uint secondSource, uint secondTarget)
231     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
232     ↪ secondTarget);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>

```

```

250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
    ↪ uint secondSource, uint secondTarget)
260     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
    ↪ secondTarget);
261
262     /// <summary>
263     /// <para>
264     /// Clears the node using the specified node.
265     /// </para>
266     /// <para></para>
267     /// </summary>
268     /// <param name="node">
269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(uint node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsSource = 0U;
277         link.RightAsSource = 0U;
278         link.SizeAsSource = 0U;
279     }
280 }
281 }

```

1.97 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see
19        ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
    ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
    ↪ header) { }
36
37        /// <summary>
38        /// <para>
39        /// Gets the left reference using the specified node.

```

```

39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref uint</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref uint</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref uint GetRightReference(uint node) => ref
69         ↪ Links[node].RightAsTarget;
70
71     /// <summary>
72     /// <para>
73     /// Gets the left using the specified node.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The uint</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;
87
88     /// <summary>
89     /// <para>
90     /// Gets the right using the specified node.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The uint</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override uint GetRight(uint node) => Links[node].RightAsTarget;
104
105    /// <summary>
106    /// <para>
107    /// Sets the left using the specified node.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="node">
112    /// <para>The node.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="left">
116    /// <para>The left.</para>

```

```

116     /// <para></para>
117     /// </param>
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]
119     protected override void SetLeft(uint node, uint left) => Links[node].LeftAsTarget = left;
120
121     /// <summary>
122     /// <para>
123     /// Sets the right using the specified node.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="node">
128     /// <para>The node.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(uint node, uint right) => Links[node].RightAsTarget =
        ↳ right;
137
138     /// <summary>
139     /// <para>
140     /// Gets the size using the specified node.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="node">
145     /// <para>The node.</para>
146     /// <para></para>
147     /// </param>
148     /// <returns>
149     /// <para>The uint</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override uint GetSize(uint node) => Links[node].SizeAsTarget;
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="node">
162     /// <para>The node.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(uint node, uint size) => Links[node].SizeAsTarget = size;
171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <returns>
179     /// <para>The uint</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override uint GetTreeRoot() => Header->RootAsTarget;
184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified link.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>

```

```

193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The uint</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override uint GetBasePartValue(uint link) => Links[link].Target;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="firstSource">
209     /// <para>The first source.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="firstTarget">
213     /// <para>The first target.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
230     ↪ uint secondSource, uint secondTarget)
231     ↪ => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
232     ↪ secondSource);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondTarget">
253     /// <para>The second target.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
262     ↪ uint secondSource, uint secondTarget)
263     ↪ => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
264     ↪ secondSource);
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>

```

```

266     /// <para></para>
267     /// </summary>
268     /// <param name="node">
269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(uint node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsTarget = 0U;
277         link.RightAsTarget = 0U;
278         link.SizeAsTarget = 0U;
279     }
280 }
281 }

```

1.98 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsSizeBalancedTreeMethods :
15         ↳ UInt32LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
36             ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38         /// <summary>
39         /// <para>
40         /// Gets the left reference using the specified node.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <param name="node">
45         /// <para>The node.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The ref uint</para>
50         /// <para></para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
54
55         /// <summary>
56         /// <para>
57         /// Gets the right reference using the specified node.
58         /// </para>
59         /// <para></para>
60         /// </summary>

```

```

59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref uint</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref uint GetRightReference(uint node) => ref
        ↳ Links[node].RightAsTarget;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The uint</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The uint</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override uint GetRight(uint node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(uint node, uint left) => Links[node].LeftAsTarget = left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

136     protected override void SetRight(uint node, uint right) => Links[node].RightAsTarget =
    ↪     right;
137
138     /// <summary>
139     /// <para>
140     /// Gets the size using the specified node.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="node">
145     /// <para>The node.</para>
146     /// <para></para>
147     /// </param>
148     /// <returns>
149     /// <para>The uint</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override uint GetSize(uint node) => Links[node].SizeAsTarget;
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="node">
162     /// <para>The node.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(uint node, uint size) => Links[node].SizeAsTarget = size;
171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <returns>
179     /// <para>The uint</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override uint GetTreeRoot() => Header->RootAsTarget;
184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified link.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The uint</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override uint GetBasePartValue(uint link) => Links[link].Target;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="firstSource">
209     /// <para>The first source.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="firstTarget">

```



```

213     /// <para>The first target.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToLeftOfSecond(uint firstSource, uint firstTarget,
230     ↪ uint secondSource, uint secondTarget)
231     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
232     ↪ secondSource);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondTarget">
253     /// <para>The second target.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
262     ↪ uint secondSource, uint secondTarget)
263     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
264     ↪ secondSource);
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>
270     /// <para></para>
271     /// </summary>
272     /// <param name="node">
273     /// <para>The node.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(uint node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsTarget = 0U;
281         link.RightAsTarget = 0U;
282         link.SizeAsTarget = 0U;
283     }
284 }

```

1.99 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Memory;

```

```

4 using Platform.Singletons;
5 using Platform.Data.Doublets.Memory.United.Generic;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ///   ↪ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
14     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
15     ///   ↪ размером, для организации хранения связей с адресами представленными в виде <see
16     ///   ↪ cref="uint"/>.</para>
17     /// </summary>
18     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
19     {
20         private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
22         private LinksHeader<uint>* _header;
23         private RawLink<uint>* _links;
24
25         /// <summary>
26         /// <para>
27         ///   Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
28         /// </para>
29         /// </summary>
30         /// <param name="address">
31         ///   <para>A address.</para>
32         /// </param>
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public UInt32UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
35
36         /// <summary>
37         ///   Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
38         ///   ↪ минимальным шагом расширения базы данных.
39         /// </summary>
40         /// <param name="address">Полный путь к файлу базы данных.</param>
41         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
42         ///   ↪ байтах.</param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
45         ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
46         ↪ memoryReservationStep) { }
47
48         /// <summary>
49         /// <para>
50         ///   Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
51         /// </para>
52         /// </summary>
53         /// <param name="memory">
54         ///   <para>A memory.</para>
55         /// </param>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
58         ↪ DefaultLinksSizeStep) { }
59
60         /// <summary>
61         /// <para>
62         ///   Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
63         /// </para>
64         /// </summary>
65         /// <param name="memory">
66         ///   <para>A memory.</para>
67         /// </param>
68         /// <param name="memoryReservationStep">
69         ///   <para>A memory reservation step.</para>
70         /// </param>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
73         ↪ memoryReservationStep) : this(memory, memoryReservationStep,
74         ↪ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }

```

```

72     /// <summary>
73     /// <para>
74     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="memory">
79     /// <para>A memory.</para>
80     /// <para></para>
81     /// </param>
82     /// <param name="memoryReservationStep">
83     /// <para>A memory reservation step.</para>
84     /// <para></para>
85     /// </param>
86     /// <param name="constants">
87     /// <para>A constants.</para>
88     /// <para></para>
89     /// </param>
90     /// <param name="indexTreeType">
91     /// <para>A index tree type.</para>
92     /// <para></para>
93     /// </param>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
96     ↪ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
97     ↪ : base(memory, memoryReservationStep, constants)
98     {
99         if (indexTreeType == IndexTreeType.SizeBalancedTree)
100         {
101             _createSourceTreeMethods = () => new
102             ↪ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
103             _createTargetTreeMethods = () => new
104             ↪ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
105         }
106         else
107         {
108             _createSourceTreeMethods = () => new
109             ↪ UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
110             ↪ _header);
111             _createTargetTreeMethods = () => new
112             ↪ UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
113             ↪ _header);
114         }
115         Init(memory, memoryReservationStep);
116     }
117     /// <summary>
118     /// <para>
119     /// Sets the pointers using the specified memory.
120     /// </para>
121     /// <para></para>
122     /// </summary>
123     /// <param name="memory">
124     /// <para>The memory.</para>
125     /// <para></para>
126     /// </param>
127     [MethodImpl(MethodImplOptions.AggressiveInlining)]
128     protected override void SetPointers(IResizableDirectMemory memory)
129     {
130         _header = (LinksHeader<uint>*)memory.Pointer;
131         _links = (RawLink<uint>*)memory.Pointer;
132         SourcesTreeMethods = _createSourceTreeMethods();
133         TargetsTreeMethods = _createTargetTreeMethods();
134         UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
135     }
136     /// <summary>
137     /// <para>
138     /// Resets the pointers.
139     /// </para>
140     /// <para></para>
141     /// </summary>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected override void ResetPointers()
144     {
145         base.ResetPointers();

```

```

141     _links = null;
142     _header = null;
143 }
144
145 /// <summary>
146 /// <para>
147 /// Gets the header reference.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <returns>
152 /// <para>A ref links header of uint</para>
153 /// <para></para>
154 /// </returns>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
157
158 /// <summary>
159 /// <para>
160 /// Gets the link reference using the specified link index.
161 /// </para>
162 /// <para></para>
163 /// </summary>
164 /// <param name="linkIndex">
165 /// <para>The link index.</para>
166 /// <para></para>
167 /// </param>
168 /// <returns>
169 /// <para>A ref raw link of uint</para>
170 /// <para></para>
171 /// </returns>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 protected override ref RawLink<uint> GetLinkReference(uint linkIndex) => ref
    ↪ _links[linkIndex];
174
175 /// <summary>
176 /// <para>
177 /// Determines whether this instance are equal.
178 /// </para>
179 /// <para></para>
180 /// </summary>
181 /// <param name="first">
182 /// <para>The first.</para>
183 /// <para></para>
184 /// </param>
185 /// <param name="second">
186 /// <para>The second.</para>
187 /// <para></para>
188 /// </param>
189 /// <returns>
190 /// <para>The bool</para>
191 /// <para></para>
192 /// </returns>
193 [MethodImpl(MethodImplOptions.AggressiveInlining)]
194 protected override bool AreEqual(uint first, uint second) => first == second;
195
196 /// <summary>
197 /// <para>
198 /// Determines whether this instance less than.
199 /// </para>
200 /// <para></para>
201 /// </summary>
202 /// <param name="first">
203 /// <para>The first.</para>
204 /// <para></para>
205 /// </param>
206 /// <param name="second">
207 /// <para>The second.</para>
208 /// <para></para>
209 /// </param>
210 /// <returns>
211 /// <para>The bool</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected override bool LessThan(uint first, uint second) => first < second;
216
217 /// <summary>

```

```

218     /// <para>
219     /// Determines whether this instance less or equal than.
220     /// </para>
221     /// <para></para>
222     /// </summary>
223     /// <param name="first">
224     /// <para>The first.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="second">
228     /// <para>The second.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool LessOrEqualThan(uint first, uint second) => first <= second;
237
238     /// <summary>
239     /// <para>
240     /// Determines whether this instance greater than.
241     /// </para>
242     /// <para></para>
243     /// </summary>
244     /// <param name="first">
245     /// <para>The first.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="second">
249     /// <para>The second.</para>
250     /// <para></para>
251     /// </param>
252     /// <returns>
253     /// <para>The bool</para>
254     /// <para></para>
255     /// </returns>
256     [MethodImpl(MethodImplOptions.AggressiveInlining)]
257     protected override bool GreaterThan(uint first, uint second) => first > second;
258
259     /// <summary>
260     /// <para>
261     /// Determines whether this instance greater or equal than.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="first">
266     /// <para>The first.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="second">
270     /// <para>The second.</para>
271     /// <para></para>
272     /// </param>
273     /// <returns>
274     /// <para>The bool</para>
275     /// <para></para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
279
280     /// <summary>
281     /// <para>
282     /// Gets the zero.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <returns>
287     /// <para>The uint</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override uint GetZero() => 0U;
292
293     /// <summary>
294     /// <para>
295     /// Gets the one.

```

```

296    /// </para>
297    /// <para></para>
298    /// </summary>
299    /// <returns>
300    /// <para>The uint</para>
301    /// <para></para>
302    /// </returns>
303    [MethodImpl(MethodImplOptions.AggressiveInlining)]
304    protected override uint GetOne() => 1U;
305
306    /// <summary>
307    /// <para>
308    /// Converts the to int 64 using the specified value.
309    /// </para>
310    /// <para></para>
311    /// </summary>
312    /// <param name="value">
313    /// <para>The value.</para>
314    /// <para></para>
315    /// </param>
316    /// <returns>
317    /// <para>The long</para>
318    /// <para></para>
319    /// </returns>
320    [MethodImpl(MethodImplOptions.AggressiveInlining)]
321    protected override long ConvertToInt64(uint value) => (long)value;
322
323    /// <summary>
324    /// <para>
325    /// Converts the to address using the specified value.
326    /// </para>
327    /// <para></para>
328    /// </summary>
329    /// <param name="value">
330    /// <para>The value.</para>
331    /// <para></para>
332    /// </param>
333    /// <returns>
334    /// <para>The uint</para>
335    /// <para></para>
336    /// </returns>
337    [MethodImpl(MethodImplOptions.AggressiveInlining)]
338    protected override uint ConvertToAddress(long value) => (uint)value;
339
340    /// <summary>
341    /// <para>
342    /// Adds the first.
343    /// </para>
344    /// <para></para>
345    /// </summary>
346    /// <param name="first">
347    /// <para>The first.</para>
348    /// <para></para>
349    /// </param>
350    /// <param name="second">
351    /// <para>The second.</para>
352    /// <para></para>
353    /// </param>
354    /// <returns>
355    /// <para>The uint</para>
356    /// <para></para>
357    /// </returns>
358    [MethodImpl(MethodImplOptions.AggressiveInlining)]
359    protected override uint Add(uint first, uint second) => first + second;
360
361    /// <summary>
362    /// <para>
363    /// Subtracts the first.
364    /// </para>
365    /// <para></para>
366    /// </summary>
367    /// <param name="first">
368    /// <para>The first.</para>
369    /// <para></para>
370    /// </param>
371    /// <param name="second">
372    /// <para>The second.</para>
373    /// <para></para>

```

```

374     /// </param>
375     /// <returns>
376     /// <para>The uint</para>
377     /// <para></para>
378     /// </returns>
379     [MethodImpl(MethodImplOptions.AggressiveInlining)]
380     protected override uint Subtract(uint first, uint second) => first - second;
381
382     /// <summary>
383     /// <para>
384     /// Increments the link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>The uint</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override uint Increment(uint link) => ++link;
398
399     /// <summary>
400     /// <para>
401     /// Decrements the link.
402     /// </para>
403     /// <para></para>
404     /// </summary>
405     /// <param name="link">
406     /// <para>The link.</para>
407     /// <para></para>
408     /// </param>
409     /// <returns>
410     /// <para>The uint</para>
411     /// <para></para>
412     /// </returns>
413     [MethodImpl(MethodImplOptions.AggressiveInlining)]
414     protected override uint Decrement(uint link) => --link;
415 }
416 }

```

1.100 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 32 unused links list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UnusedLinksListMethods{uint}"/>
15     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<uint>
16     {
17         private readonly RawLink<uint>* _links;
18         private readonly LinksHeader<uint>* _header;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

35     public UInt32UnusedLinksListMethods(RawLink<uint>* links, LinksHeader<uint>* header)
36     : base((byte*)links, (byte*)header)
37     {
38         _links = links;
39         _header = header;
40     }
41
42     /// <summary>
43     /// <para>
44     /// Gets the link reference using the specified link.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="link">
49     /// <para>The link.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>A ref raw link of uint</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref RawLink<uint> GetLinkReference(uint link) => ref _links[link];
58
59     /// <summary>
60     /// <para>
61     /// Gets the header reference.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>A ref links header of uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
71 }
72 }

```

1.101 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using static System.Runtime.CompilerServices.Unsafe;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.United.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 links avl balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{ulong}"/>
16     public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
17         ↳ LinksAvlBalancedTreeMethodsBase<ulong>
18     {
19         /// <summary>
20         /// <para>
21         /// The links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLink<ulong>* Links;
26
27         /// <summary>
28         /// <para>
29         /// The header.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly LinksHeader<ulong>* Header;
34
35         /// <summary>
36         /// <para>
37         /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase"/> instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>

```



```

39     /// <param name="constants">
40     /// <para>A constants.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="links">
44     /// <para>A links.</para>
45     /// <para></para>
46     /// </param>
47     /// <param name="header">
48     /// <para>A header.</para>
49     /// <para></para>
50     /// </param>
51     protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
52     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
53         : base(constants, (byte*)links, (byte*)header)
54     {
55         Links = links;
56         Header = header;
57     }
58     /// <summary>
59     /// <para>
60     /// Gets the zero.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <returns>
65     /// <para>The ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ulong GetZero() => OUL;
70
71     /// <summary>
72     /// <para>
73     /// Determines whether this instance equal to zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="value">
78     /// <para>The value.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The bool</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override bool EqualToZero(ulong value) => value == OUL;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance are equal.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="first">
95     /// <para>The first.</para>
96     /// <para></para>
97     /// </param>
98     /// <param name="second">
99     /// <para>The second.</para>
100    /// <para></para>
101    /// </param>
102    /// <returns>
103    /// <para>The bool</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override bool AreEqual(ulong first, ulong second) => first == second;
108
109    /// <summary>
110    /// <para>
111    /// Determines whether this instance greater than zero.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="value">

```

```

116    /// <para>The value.</para>
117    /// <para></para>
118    /// </param>
119    /// <returns>
120    /// <para>The bool</para>
121    /// <para></para>
122    /// </returns>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override bool GreaterThanZero(ulong value) => value > 0UL;
125
126    /// <summary>
127    /// <para>
128    /// Determines whether this instance greater than.
129    /// </para>
130    /// <para></para>
131    /// </summary>
132    /// <param name="first">
133    /// <para>The first.</para>
134    /// <para></para>
135    /// </param>
136    /// <param name="second">
137    /// <para>The second.</para>
138    /// <para></para>
139    /// </param>
140    /// <returns>
141    /// <para>The bool</para>
142    /// <para></para>
143    /// </returns>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override bool GreaterThan(ulong first, ulong second) => first > second;
146
147    /// <summary>
148    /// <para>
149    /// Determines whether this instance greater or equal than.
150    /// </para>
151    /// <para></para>
152    /// </summary>
153    /// <param name="first">
154    /// <para>The first.</para>
155    /// <para></para>
156    /// </param>
157    /// <param name="second">
158    /// <para>The second.</para>
159    /// <para></para>
160    /// </param>
161    /// <returns>
162    /// <para>The bool</para>
163    /// <para></para>
164    /// </returns>
165    [MethodImpl(MethodImplOptions.AggressiveInlining)]
166    protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
167
168    /// <summary>
169    /// <para>
170    /// Determines whether this instance greater or equal than zero.
171    /// </para>
172    /// <para></para>
173    /// </summary>
174    /// <param name="value">
175    /// <para>The value.</para>
176    /// <para></para>
177    /// </param>
178    /// <returns>
179    /// <para>The bool</para>
180    /// <para></para>
181    /// </returns>
182    [MethodImpl(MethodImplOptions.AggressiveInlining)]
183    protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ always true for ulong
184
185    /// <summary>
186    /// <para>
187    /// Determines whether this instance less or equal than zero.
188    /// </para>
189    /// <para></para>
190    /// </summary>
191    /// <param name="value">
192    /// <para>The value.</para>

```

```

193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The bool</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance less or equal than.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="first">
209     /// <para>The first.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="second">
213     /// <para>The second.</para>
214     /// <para></para>
215     /// </param>
216     /// <returns>
217     /// <para>The bool</para>
218     /// <para></para>
219     /// </returns>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
222
223     /// <summary>
224     /// <para>
225     /// Determines whether this instance less than zero.
226     /// </para>
227     /// <para></para>
228     /// </summary>
229     /// <param name="value">
230     /// <para>The value.</para>
231     /// <para></para>
232     /// </param>
233     /// <returns>
234     /// <para>The bool</para>
235     /// <para></para>
236     /// </returns>
237     [MethodImpl(MethodImplOptions.AggressiveInlining)]
238     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance less than.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="first">
247     /// <para>The first.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="second">
251     /// <para>The second.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool LessThan(ulong first, ulong second) => first < second;
260
261     /// <summary>
262     /// <para>
263     /// Increments the value.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="value">
268     /// <para>The value.</para>

```

```

269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The ulong</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override ulong Increment(ulong value) => ++value;
277
278     /// <summary>
279     /// <para>
280     /// Decrements the value.
281     /// </para>
282     /// <para></para>
283     /// </summary>
284     /// <param name="value">
285     /// <para>The value.</para>
286     /// <para></para>
287     /// </param>
288     /// <returns>
289     /// <para>The ulong</para>
290     /// <para></para>
291     /// </returns>
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     protected override ulong Decrement(ulong value) => --value;
294
295     /// <summary>
296     /// <para>
297     /// Adds the first.
298     /// </para>
299     /// <para></para>
300     /// </summary>
301     /// <param name="first">
302     /// <para>The first.</para>
303     /// <para></para>
304     /// </param>
305     /// <param name="second">
306     /// <para>The second.</para>
307     /// <para></para>
308     /// </param>
309     /// <returns>
310     /// <para>The ulong</para>
311     /// <para></para>
312     /// </returns>
313     [MethodImpl(MethodImplOptions.AggressiveInlining)]
314     protected override ulong Add(ulong first, ulong second) => first + second;
315
316     /// <summary>
317     /// <para>
318     /// Subtracts the first.
319     /// </para>
320     /// <para></para>
321     /// </summary>
322     /// <param name="first">
323     /// <para>The first.</para>
324     /// <para></para>
325     /// </param>
326     /// <param name="second">
327     /// <para>The second.</para>
328     /// <para></para>
329     /// </param>
330     /// <returns>
331     /// <para>The ulong</para>
332     /// <para></para>
333     /// </returns>
334     [MethodImpl(MethodImplOptions.AggressiveInlining)]
335     protected override ulong Subtract(ulong first, ulong second) => first - second;
336
337     /// <summary>
338     /// <para>
339     /// Determines whether this instance first is to the left of second.
340     /// </para>
341     /// <para></para>
342     /// </summary>
343     /// <param name="first">
344     /// <para>The first.</para>
345     /// <para></para>
346     /// </param>

```

```

347     /// <param name="second">
348     /// <para>The second.</para>
349     /// <para></para>
350     /// </param>
351     /// <returns>
352     /// <para>The bool</para>
353     /// <para></para>
354     /// </returns>
355     [MethodImpl(MethodImplOptions.AggressiveInlining)]
356     protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
357     {
358         ref var firstLink = ref Links[first];
359         ref var secondLink = ref Links[second];
360         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
361             ↪ secondLink.Source, secondLink.Target);
362     }
363     /// <summary>
364     /// <para>
365     /// Determines whether this instance first is to the right of second.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="first">
370     /// <para>The first.</para>
371     /// <para></para>
372     /// </param>
373     /// <param name="second">
374     /// <para>The second.</para>
375     /// <para></para>
376     /// </param>
377     /// <returns>
378     /// <para>The bool</para>
379     /// <para></para>
380     /// </returns>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383     {
384         ref var firstLink = ref Links[first];
385         ref var secondLink = ref Links[second];
386         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387             ↪ secondLink.Source, secondLink.Target);
388     }
389     /// <summary>
390     /// <para>
391     /// Gets the size value using the specified value.
392     /// </para>
393     /// <para></para>
394     /// </summary>
395     /// <param name="value">
396     /// <para>The value.</para>
397     /// <para></para>
398     /// </param>
399     /// <returns>
400     /// <para>The ulong</para>
401     /// <para></para>
402     /// </returns>
403     [MethodImpl(MethodImplOptions.AggressiveInlining)]
404     protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406     /// <summary>
407     /// <para>
408     /// Sets the size value using the specified stored value.
409     /// </para>
410     /// <para></para>
411     /// </summary>
412     /// <param name="storedValue">
413     /// <para>The stored value.</para>
414     /// <para></para>
415     /// </param>
416     /// <param name="size">
417     /// <para>The size.</para>
418     /// <para></para>
419     /// </param>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
422         ↪ storedValue & 31UL | (size & 134217727UL) << 5;

```

```

422     /// <summary>
423     /// <para>
424     /// Determines whether this instance get left is child value.
425     /// </para>
426     /// <para></para>
427     /// </summary>
428     /// <param name="value">
429     /// <para>The value.</para>
430     /// <para></para>
431     /// </param>
432     /// <returns>
433     /// <para>The bool</para>
434     /// <para></para>
435     /// </returns>
436     [MethodImpl(MethodImplOptions.AggressiveInlining)]
437     protected override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;
438
439     /// <summary>
440     /// <para>
441     /// Sets the left is child value using the specified stored value.
442     /// </para>
443     /// <para></para>
444     /// </summary>
445     /// <param name="storedValue">
446     /// <para>The stored value.</para>
447     /// <para></para>
448     /// </param>
449     /// <param name="value">
450     /// <para>The value.</para>
451     /// <para></para>
452     /// </param>
453     [MethodImpl(MethodImplOptions.AggressiveInlining)]
454     protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
455     ↪ storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;
456
457     /// <summary>
458     /// <para>
459     /// Determines whether this instance get right is child value.
460     /// </para>
461     /// <para></para>
462     /// </summary>
463     /// <param name="value">
464     /// <para>The value.</para>
465     /// <para></para>
466     /// </param>
467     /// <returns>
468     /// <para>The bool</para>
469     /// <para></para>
470     /// </returns>
471     [MethodImpl(MethodImplOptions.AggressiveInlining)]
472     protected override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;
473
474     /// <summary>
475     /// <para>
476     /// Sets the right is child value using the specified stored value.
477     /// </para>
478     /// <para></para>
479     /// </summary>
480     /// <param name="storedValue">
481     /// <para>The stored value.</para>
482     /// <para></para>
483     /// </param>
484     /// <param name="value">
485     /// <para>The value.</para>
486     /// <para></para>
487     /// </param>
488     [MethodImpl(MethodImplOptions.AggressiveInlining)]
489     protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
490     ↪ storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;
491
492     /// <summary>
493     /// <para>
494     /// Gets the balance value using the specified value.
495     /// </para>
496     /// <para></para>
497     /// </summary>
498     /// <param name="value">

```

```

498     /// <para>The value.</para>
499     /// <para></para>
500     /// </param>
501     /// <returns>
502     /// <para>The sbyte</para>
503     /// <para></para>
504     /// </returns>
505     [MethodImpl(MethodImplOptions.AggressiveInlining)]
506     protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
    ↪ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
    ↪ sbyte

507     /// <summary>
508     /// <para>
509     /// Sets the balance value using the specified stored value.
510     /// </para>
511     /// <para></para>
512     /// </summary>
513     /// <param name="storedValue">
514     /// <para>The stored value.</para>
515     /// <para></para>
516     /// </param>
517     /// <param name="value">
518     /// <para>The value.</para>
519     /// <para></para>
520     /// </param>
521     [MethodImpl(MethodImplOptions.AggressiveInlining)]
522     protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
    ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
    ↪ value & 3) & 7UL);

524     /// <summary>
525     /// <para>
526     /// Gets the header reference.
527     /// </para>
528     /// <para></para>
529     /// </summary>
530     /// <returns>
531     /// <para>A ref links header of ulong</para>
532     /// <para></para>
533     /// </returns>
534     [MethodImpl(MethodImplOptions.AggressiveInlining)]
535     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;

538     /// <summary>
539     /// <para>
540     /// Gets the link reference using the specified link.
541     /// </para>
542     /// <para></para>
543     /// </summary>
544     /// <param name="link">
545     /// <para>The link.</para>
546     /// <para></para>
547     /// </param>
548     /// <returns>
549     /// <para>A ref raw link of ulong</para>
550     /// <para></para>
551     /// </returns>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
554 }
555 }

```

1.102 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMeth

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 64 links recursionless size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{ulong}">

```

```

15 public unsafe abstract class UInt64LinksRecursionlessSizeBalancedTreeMethodsBase :
    ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<ulong>
16 {
17     /// <summary>
18     /// <para>
19     /// The links.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     protected new readonly RawLink<ulong>* Links;
24     /// <summary>
25     /// <para>
26     /// The header.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     protected new readonly LinksHeader<ulong>* Header;
31
32     /// <summary>
33     /// <para>
34     ↳ Initializes a new <see cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
35     instance.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="constants">
40     /// <para>A constants.</para>
41     /// </param>
42     /// <param name="links">
43     /// <para>A links.</para>
44     /// </param>
45     /// <param name="header">
46     /// <para>A header.</para>
47     /// </param>
48     protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
    ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
49     : base(constants, (byte*)links, (byte*)header)
50     {
51         Links = links;
52         Header = header;
53     }
54
55     /// <summary>
56     /// <para>
57     /// Gets the zero.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <returns>
62     /// <para>The ulong</para>
63     /// <para></para>
64     /// </returns>
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     protected override ulong GetZero() => OUL;
67
68     /// <summary>
69     /// <para>
70     /// Determines whether this instance equal to zero.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="value">
75     /// <para>The value.</para>
76     /// </param>
77     /// <returns>
78     /// <para>The bool</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override bool EqualToZero(ulong value) => value == OUL;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance are equal.
87     /// </para>
88     /// </summary>
89     /// <para>

```



```

90    /// </para>
91    /// <para></para>
92    /// </summary>
93    /// <param name="first">
94    /// <para>The first.</para>
95    /// <para></para>
96    /// </param>
97    /// <param name="second">
98    /// <para>The second.</para>
99    /// <para></para>
100   /// </param>
101   /// <returns>
102   /// <para>The bool</para>
103   /// <para></para>
104   /// </returns>
105   [MethodImpl(MethodImplOptions.AggressiveInlining)]
106   protected override bool AreEqual(ulong first, ulong second) => first == second;
107
108   /// <summary>
109   /// <para>
110   /// Determines whether this instance greater than zero.
111   /// </para>
112   /// <para></para>
113   /// </summary>
114   /// <param name="value">
115   /// <para>The value.</para>
116   /// <para></para>
117   /// </param>
118   /// <returns>
119   /// <para>The bool</para>
120   /// <para></para>
121   /// </returns>
122   [MethodImpl(MethodImplOptions.AggressiveInlining)]
123   protected override bool GreaterThanZero(ulong value) => value > 0UL;
124
125   /// <summary>
126   /// <para>
127   /// Determines whether this instance greater than.
128   /// </para>
129   /// <para></para>
130   /// </summary>
131   /// <param name="first">
132   /// <para>The first.</para>
133   /// <para></para>
134   /// </param>
135   /// <param name="second">
136   /// <para>The second.</para>
137   /// <para></para>
138   /// </param>
139   /// <returns>
140   /// <para>The bool</para>
141   /// <para></para>
142   /// </returns>
143   [MethodImpl(MethodImplOptions.AggressiveInlining)]
144   protected override bool GreaterThan(ulong first, ulong second) => first > second;
145
146   /// <summary>
147   /// <para>
148   /// Determines whether this instance greater or equal than.
149   /// </para>
150   /// <para></para>
151   /// </summary>
152   /// <param name="first">
153   /// <para>The first.</para>
154   /// <para></para>
155   /// </param>
156   /// <param name="second">
157   /// <para>The second.</para>
158   /// <para></para>
159   /// </param>
160   /// <returns>
161   /// <para>The bool</para>
162   /// <para></para>
163   /// </returns>
164   [MethodImpl(MethodImplOptions.AggressiveInlining)]
165   protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
166
167   /// <summary>

```

```

168     /// <para>
169     /// Determines whether this instance greater or equal than zero.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="value">
174     /// <para>The value.</para>
175     /// <para></para>
176     /// </param>
177     /// <returns>
178     /// <para>The bool</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ always true for ulong
183
184     /// <summary>
185     /// <para>
186     /// Determines whether this instance less or equal than zero.
187     /// </para>
188     /// <para></para>
189     /// </summary>
190     /// <param name="value">
191     /// <para>The value.</para>
192     /// <para></para>
193     /// </param>
194     /// <returns>
195     /// <para>The bool</para>
196     /// <para></para>
197     /// </returns>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance less or equal than.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="first">
208     /// <para>The first.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="second">
212     /// <para>The second.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>The bool</para>
217     /// <para></para>
218     /// </returns>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
221
222     /// <summary>
223     /// <para>
224     /// Determines whether this instance less than zero.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="value">
229     /// <para>The value.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong
238
239     /// <summary>
240     /// <para>
241     /// Determines whether this instance less than.
242     /// </para>

```

```

243     /// <para></para>
244     /// </summary>
245     /// <param name="first">
246     /// <para>The first.</para>
247     /// <para></para>
248     /// </param>
249     /// <param name="second">
250     /// <para>The second.</para>
251     /// <para></para>
252     /// </param>
253     /// <returns>
254     /// <para>The bool</para>
255     /// <para></para>
256     /// </returns>
257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
258     protected override bool LessThan(ulong first, ulong second) => first < second;
259
260     /// <summary>
261     /// <para>
262     /// Increments the value.
263     /// </para>
264     /// <para></para>
265     /// </summary>
266     /// <param name="value">
267     /// <para>The value.</para>
268     /// <para></para>
269     /// </param>
270     /// <returns>
271     /// <para>The ulong</para>
272     /// <para></para>
273     /// </returns>
274     [MethodImpl(MethodImplOptions.AggressiveInlining)]
275     protected override ulong Increment(ulong value) => ++value;
276
277     /// <summary>
278     /// <para>
279     /// Decrements the value.
280     /// </para>
281     /// <para></para>
282     /// </summary>
283     /// <param name="value">
284     /// <para>The value.</para>
285     /// <para></para>
286     /// </param>
287     /// <returns>
288     /// <para>The ulong</para>
289     /// <para></para>
290     /// </returns>
291     [MethodImpl(MethodImplOptions.AggressiveInlining)]
292     protected override ulong Decrement(ulong value) => --value;
293
294     /// <summary>
295     /// <para>
296     /// Adds the first.
297     /// </para>
298     /// <para></para>
299     /// </summary>
300     /// <param name="first">
301     /// <para>The first.</para>
302     /// <para></para>
303     /// </param>
304     /// <param name="second">
305     /// <para>The second.</para>
306     /// <para></para>
307     /// </param>
308     /// <returns>
309     /// <para>The ulong</para>
310     /// <para></para>
311     /// </returns>
312     [MethodImpl(MethodImplOptions.AggressiveInlining)]
313     protected override ulong Add(ulong first, ulong second) => first + second;
314
315     /// <summary>
316     /// <para>
317     /// Subtracts the first.
318     /// </para>
319     /// <para></para>
320     /// </summary>

```

```

321    /// <param name="first">
322    /// <para>The first.</para>
323    /// <para></para>
324    /// </param>
325    /// <param name="second">
326    /// <para>The second.</para>
327    /// <para></para>
328    /// </param>
329    /// <returns>
330    /// <para>The ulong</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override ulong Subtract(ulong first, ulong second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>
348    /// <para></para>
349    /// </param>
350    /// <returns>
351    /// <para>The bool</para>
352    /// <para></para>
353    /// </returns>
354    [MethodImpl(MethodImplOptions.AggressiveInlining)]
355    protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }
362
363    /// <summary>
364    /// <para>
365    /// Determines whether this instance first is to the right of second.
366    /// </para>
367    /// <para></para>
368    /// </summary>
369    /// <param name="first">
370    /// <para>The first.</para>
371    /// <para></para>
372    /// </param>
373    /// <param name="second">
374    /// <para>The second.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383    {
384        ref var firstLink = ref Links[first];
385        ref var secondLink = ref Links[second];
386        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387            ↪ secondLink.Source, secondLink.Target);
388    }
389
390    /// <summary>
391    /// <para>
392    /// Gets the header reference.
393    /// </para>
394    /// <para></para>
395    /// </summary>
396    /// <returns>
397    /// <para>A ref links header of ulong</para>
398    /// <para></para>

```

```

397     /// </returns>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
400
401     /// <summary>
402     /// <para>
403     /// Gets the link reference using the specified link.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     /// <param name="link">
408     /// <para>The link.</para>
409     /// <para></para>
410     /// </param>
411     /// <returns>
412     /// <para>A ref raw link of ulong</para>
413     /// <para></para>
414     /// </returns>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
417 }
418 }

```

1.103 ./csharp/Platform.Data.Doublets.Memory.United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 links size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{ulong}" />
15     public unsafe abstract class UInt64LinksSizeBalancedTreeMethodsBase :
16     ↳ LinksSizeBalancedTreeMethodsBase<ulong>
17     {
18         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLink<ulong>* Links;
25
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<ulong>* Header;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase" /> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">
41         /// <para>A constants.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="links">
45         /// <para>A links.</para>
46         /// <para></para>
47         /// </param>
48         /// <param name="header">
49         /// <para>A header.</para>
50         /// <para></para>
51         /// </param>
52         protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
53         ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
54         : base(constants, (byte*)links, (byte*)header)
55     {
56

```

```

53     Links = links;
54     Header = header;
55 }
56
57 /// <summary>
58 /// <para>
59 /// Gets the zero.
60 /// </para>
61 /// <para></para>
62 /// </summary>
63 /// <returns>
64 /// <para>The ulong</para>
65 /// <para></para>
66 /// </returns>
67 [MethodImpl(MethodImplOptions.AggressiveInlining)]
68 protected override ulong GetZero() => OUL;
69
70 /// <summary>
71 /// <para>
72 /// Determines whether this instance equal to zero.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 /// <param name="value">
77 /// <para>The value.</para>
78 /// <para></para>
79 /// </param>
80 /// <returns>
81 /// <para>The bool</para>
82 /// <para></para>
83 /// </returns>
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 protected override bool EqualToZero(ulong value) => value == OUL;
86
87 /// <summary>
88 /// <para>
89 /// Determines whether this instance are equal.
90 /// </para>
91 /// <para></para>
92 /// </summary>
93 /// <param name="first">
94 /// <para>The first.</para>
95 /// <para></para>
96 /// </param>
97 /// <param name="second">
98 /// <para>The second.</para>
99 /// <para></para>
100 /// </param>
101 /// <returns>
102 /// <para>The bool</para>
103 /// <para></para>
104 /// </returns>
105 [MethodImpl(MethodImplOptions.AggressiveInlining)]
106 protected override bool AreEqual(ulong first, ulong second) => first == second;
107
108 /// <summary>
109 /// <para>
110 /// Determines whether this instance greater than zero.
111 /// </para>
112 /// <para></para>
113 /// </summary>
114 /// <param name="value">
115 /// <para>The value.</para>
116 /// <para></para>
117 /// </param>
118 /// <returns>
119 /// <para>The bool</para>
120 /// <para></para>
121 /// </returns>
122 [MethodImpl(MethodImplOptions.AggressiveInlining)]
123 protected override bool GreaterThanZero(ulong value) => value > OUL;
124
125 /// <summary>
126 /// <para>
127 /// Determines whether this instance greater than.
128 /// </para>
129 /// <para></para>
130 /// </summary>

```

```

131     /// <param name="first">
132     /// <para>The first.</para>
133     /// <para></para>
134     /// </param>
135     /// <param name="second">
136     /// <para>The second.</para>
137     /// <para></para>
138     /// </param>
139     /// <returns>
140     /// <para>The bool</para>
141     /// <para></para>
142     /// </returns>
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     protected override bool GreaterThan(ulong first, ulong second) => first > second;
145
146     /// <summary>
147     /// <para>
148     /// Determines whether this instance greater or equal than.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="first">
153     /// <para>The first.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="second">
157     /// <para>The second.</para>
158     /// <para></para>
159     /// </param>
160     /// <returns>
161     /// <para>The bool</para>
162     /// <para></para>
163     /// </returns>
164     [MethodImpl(MethodImplOptions.AggressiveInlining)]
165     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
166
167     /// <summary>
168     /// <para>
169     /// Determines whether this instance greater or equal than zero.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="value">
174     /// <para>The value.</para>
175     /// <para></para>
176     /// </param>
177     /// <returns>
178     /// <para>The bool</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
183     ↪ always true for ulong
184
185     /// <summary>
186     /// <para>
187     /// Determines whether this instance less or equal than zero.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="value">
192     /// <para>The value.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The bool</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
201     ↪ always >= 0 for ulong
202
203     /// <summary>
204     /// <para>
205     /// Determines whether this instance less or equal than.
206     /// </para>
207     /// <para></para>
208     /// </summary>

```

```

207     /// <param name="first">
208     /// <para>The first.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="second">
212     /// <para>The second.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>The bool</para>
217     /// <para></para>
218     /// </returns>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
221
222     /// <summary>
223     /// <para>
224     /// Determines whether this instance less than zero.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="value">
229     /// <para>The value.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
238     ↪ for ulong
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance less than.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="first">
247     /// <para>The first.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="second">
251     /// <para>The second.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool LessThan(ulong first, ulong second) => first < second;
260
261     /// <summary>
262     /// <para>
263     /// Increments the value.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="value">
268     /// <para>The value.</para>
269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The ulong</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override ulong Increment(ulong value) => ++value;
277
278     /// <summary>
279     /// <para>
280     /// Decrements the value.
281     /// </para>
282     /// <para></para>
283     /// </summary>
284     /// <param name="value">

```



```

284    /// <para>The value.</para>
285    /// <para></para>
286    /// </param>
287    /// <returns>
288    /// <para>The ulong</para>
289    /// <para></para>
290    /// </returns>
291    [MethodImpl(MethodImplOptions.AggressiveInlining)]
292    protected override ulong Decrement(ulong value) => --value;
293
294    /// <summary>
295    /// <para>
296    /// Adds the first.
297    /// </para>
298    /// <para></para>
299    /// </summary>
300    /// <param name="first">
301    /// <para>The first.</para>
302    /// <para></para>
303    /// </param>
304    /// <param name="second">
305    /// <para>The second.</para>
306    /// <para></para>
307    /// </param>
308    /// <returns>
309    /// <para>The ulong</para>
310    /// <para></para>
311    /// </returns>
312    [MethodImpl(MethodImplOptions.AggressiveInlining)]
313    protected override ulong Add(ulong first, ulong second) => first + second;
314
315    /// <summary>
316    /// <para>
317    /// Subtracts the first.
318    /// </para>
319    /// <para></para>
320    /// </summary>
321    /// <param name="first">
322    /// <para>The first.</para>
323    /// <para></para>
324    /// </param>
325    /// <param name="second">
326    /// <para>The second.</para>
327    /// <para></para>
328    /// </param>
329    /// <returns>
330    /// <para>The ulong</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override ulong Subtract(ulong first, ulong second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>
348    /// <para></para>
349    /// </param>
350    /// <returns>
351    /// <para>The bool</para>
352    /// <para></para>
353    /// </returns>
354    [MethodImpl(MethodImplOptions.AggressiveInlining)]
355    protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }

```

```

361     /// <summary>
362     /// <para>
363     /// Determines whether this instance first is to the right of second.
364     /// </para>
365     /// </summary>
366     /// <param name="first">
367     /// <para>The first.</para>
368     /// </param>
369     /// <param name="second">
370     /// <para>The second.</para>
371     /// </param>
372     /// <returns>
373     /// <para>The bool</para>
374     /// </returns>
375     [MethodImpl(MethodImplOptions.AggressiveInlining)]
376     protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
377     {
378         ref var firstLink = ref Links[first];
379         ref var secondLink = ref Links[second];
380         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
381             ↪ secondLink.Source, secondLink.Target);
382     }
383
384     /// <summary>
385     /// <para>
386     /// Gets the header reference.
387     /// </para>
388     /// </summary>
389     /// <returns>
390     /// <para>A ref links header of ulong</para>
391     /// </returns>
392     [MethodImpl(MethodImplOptions.AggressiveInlining)]
393     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
394
395     /// <summary>
396     /// <para>
397     /// Gets the link reference using the specified link.
398     /// </para>
399     /// </summary>
400     /// <param name="link">
401     /// <para>The link.</para>
402     /// </param>
403     /// <returns>
404     /// <para>A ref raw link of ulong</para>
405     /// </returns>
406     [MethodImpl(MethodImplOptions.AggressiveInlining)]
407     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
408 }
409
410 }
411
412 }

```

1.104 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links sources avl balanced tree methods.
10    /// </para>
11    /// </summary>
12    /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
13    public unsafe class UInt64LinksSourcesAvlBalancedTreeMethods :
14        ↪ UInt64LinksAvlBalancedTreeMethodsBase
15    {
16        /// <summary>

```

```

17     /// <para>
18     /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
35     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
36     ↪ { }
37
38     /// <summary>
39     /// <para>
40     /// Gets the left reference using the specified node.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     /// <param name="node">
45     /// <para>The node.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The ref ulong</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override ref ulong GetLeftReference(ulong node) => ref
54     ↪ Links[node].LeftAsSource;
55
56     /// <summary>
57     /// <para>
58     /// Gets the right reference using the specified node.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="node">
63     /// <para>The node.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The ref ulong</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref ulong GetRightReference(ulong node) => ref
72     ↪ Links[node].RightAsSource;
73
74     /// <summary>
75     /// <para>
76     /// Gets the left using the specified node.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="node">
81     /// <para>The node.</para>
82     /// <para></para>
83     /// </param>
84     /// <returns>
85     /// <para>The ulong</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
90
91     /// <summary>
92     /// <para>
93     /// Gets the right using the specified node.
94     /// </para>

```

```

91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
120        ↪ left;
121
122    /// <summary>
123    /// <para>
124    /// Sets the right using the specified node.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="node">
129    /// <para>The node.</para>
130    /// <para></para>
131    /// </param>
132    /// <param name="right">
133    /// <para>The right.</para>
134    /// <para></para>
135    /// </param>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
138        ↪ right;
139
140    /// <summary>
141    /// <para>
142    /// Gets the size using the specified node.
143    /// </para>
144    /// <para></para>
145    /// </summary>
146    /// <param name="node">
147    /// <para>The node.</para>
148    /// <para></para>
149    /// </param>
150    /// <returns>
151    /// <para>The ulong</para>
152    /// <para></para>
153    /// </returns>
154    [MethodImpl(MethodImplOptions.AggressiveInlining)]
155    protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsSource);
156
157    /// <summary>
158    /// <para>
159    /// Sets the size using the specified node.
160    /// </para>
161    /// <para></para>
162    /// </summary>
163    /// <param name="node">
164    /// <para>The node.</para>
165    /// <para></para>
166    /// </param>
167    /// <param name="size">
168    /// <para>The size.</para>

```

```

167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
        ↳ Links[node].SizeAsSource, size);

171     /// <summary>
172     /// <para>
173     /// Determines whether this instance get left is child.
174     /// </para>
175     /// <para></para>
176     /// </summary>
177     /// <param name="node">
178     /// <para>The node.</para>
179     /// <para></para>
180     /// </param>
181     /// <returns>
182     /// <para>The bool</para>
183     /// <para></para>
184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected override bool GetLeftIsChild(ulong node) =>
187         ↳ GetLeftIsChildValue(Links[node].SizeAsSource);

188     ///[MethodImpl(MethodImplOptions.AggressiveInlining)]
189     ///protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));

191     /// <summary>
192     /// <para>
193     /// Sets the left is child using the specified node.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="node">
198     /// <para>The node.</para>
199     /// <para></para>
200     /// </param>
201     /// <param name="value">
202     /// <para>The value.</para>
203     /// <para></para>
204     /// </param>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override void SetLeftIsChild(ulong node, bool value) =>
207         ↳ SetLeftIsChildValue(ref Links[node].SizeAsSource, value);

208     /// <summary>
209     /// <para>
210     /// Determines whether this instance get right is child.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="node">
215     /// <para>The node.</para>
216     /// <para></para>
217     /// </param>
218     /// <returns>
219     /// <para>The bool</para>
220     /// <para></para>
221     /// </returns>
222     [MethodImpl(MethodImplOptions.AggressiveInlining)]
223     protected override bool GetRightIsChild(ulong node) =>
224         ↳ GetRightIsChildValue(Links[node].SizeAsSource);

225     ///[MethodImpl(MethodImplOptions.AggressiveInlining)]
226     ///protected override bool GetRightIsChild(ulong node) => IsChild(node, GetRight(node));

228     /// <summary>
229     /// <para>
230     /// Sets the right is child using the specified node.
231     /// </para>
232     /// <para></para>
233     /// </summary>
234     /// <param name="node">
235     /// <para>The node.</para>
236     /// <para></para>
237     /// </param>
238     /// <param name="value">
239     /// <para>The value.</para>
240     /// <para></para>

```

```

241 /// <para></para>
242 /// </param>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 protected override void SetRightIsChild(ulong node, bool value) =>
245     ↳ SetRightIsChildValue(ref Links[node].SizeAsSource, value);
246
247 /// <summary>
248 /// <para>
249 /// Gets the balance using the specified node.
250 /// </para>
251 /// <para></para>
252 /// </summary>
253 /// <param name="node">
254 /// <para>The node.</para>
255 /// </param>
256 /// <returns>
257 /// <para>The sbyte</para>
258 /// <para></para>
259 /// </returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 protected override sbyte GetBalance(ulong node) =>
262     ↳ GetBalanceValue(Links[node].SizeAsSource);
263
264 /// <summary>
265 /// <para>
266 /// Sets the balance using the specified node.
267 /// </para>
268 /// <para></para>
269 /// </summary>
270 /// <param name="node">
271 /// <para>The node.</para>
272 /// </param>
273 /// <param name="value">
274 /// <para>The value.</para>
275 /// </param>
276 /// </param>
277 [MethodImpl(MethodImplOptions.AggressiveInlining)]
278 protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
279     ↳ Links[node].SizeAsSource, value);
280
281 /// <summary>
282 /// <para>
283 /// Gets the tree root.
284 /// </para>
285 /// <para></para>
286 /// </summary>
287 /// <returns>
288 /// <para>The ulong</para>
289 /// <para></para>
290 /// </returns>
291 [MethodImpl(MethodImplOptions.AggressiveInlining)]
292 protected override ulong GetTreeRoot() => Header->RootAsSource;
293
294 /// <summary>
295 /// <para>
296 /// Gets the base part value using the specified link.
297 /// </para>
298 /// <para></para>
299 /// </summary>
300 /// <param name="link">
301 /// <para>The link.</para>
302 /// </param>
303 /// <returns>
304 /// <para>The ulong</para>
305 /// <para></para>
306 /// </returns>
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 protected override ulong GetBasePartValue(ulong link) => Links[link].Source;
309
310 /// <summary>
311 /// <para>
312 /// Determines whether this instance first is to the left of second.
313 /// </para>
314 /// <para></para>
315 /// </summary>

```

```

316     /// <param name="firstSource">
317     /// <para>The first source.</para>
318     /// <para></para>
319     /// </param>
320     /// <param name="firstTarget">
321     /// <para>The first target.</para>
322     /// <para></para>
323     /// </param>
324     /// <param name="secondSource">
325     /// <para>The second source.</para>
326     /// <para></para>
327     /// </param>
328     /// <param name="secondTarget">
329     /// <para>The second target.</para>
330     /// <para></para>
331     /// </param>
332     /// <returns>
333     /// <para>The bool</para>
334     /// <para></para>
335     /// </returns>
336     [MethodImpl(MethodImplOptions.AggressiveInlining)]
337     protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
338     ↪     ulong secondSource, ulong secondTarget)
339     ↪     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
340     ↪     secondTarget);
341
342     /// <summary>
343     /// <para>
344     /// Determines whether this instance first is to the right of second.
345     /// </para>
346     /// <para></para>
347     /// </summary>
348     /// <param name="firstSource">
349     /// <para>The first source.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="firstTarget">
353     /// <para>The first target.</para>
354     /// <para></para>
355     /// </param>
356     /// <param name="secondSource">
357     /// <para>The second source.</para>
358     /// <para></para>
359     /// </param>
360     /// <param name="secondTarget">
361     /// <para>The second target.</para>
362     /// <para></para>
363     /// </param>
364     /// <returns>
365     /// <para>The bool</para>
366     /// <para></para>
367     /// </returns>
368     [MethodImpl(MethodImplOptions.AggressiveInlining)]
369     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
370     ↪     ulong secondSource, ulong secondTarget)
371     ↪     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
372     ↪     secondTarget);
373
374     /// <summary>
375     /// <para>
376     /// Clears the node using the specified node.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <param name="node">
381     /// <para>The node.</para>
382     /// <para></para>
383     /// </param>
384     [MethodImpl(MethodImplOptions.AggressiveInlining)]
385     protected override void ClearNode(ulong node)
386     {
387         ref var link = ref Links[node];
388         link.LeftAsSource = OUL;
389         link.RightAsSource = OUL;
390         link.SizeAsSource = OUL;
391     }
392 }
393 }

```

1.105 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethodsBase.cs

```
1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods :
15        ↳ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↳ cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>
36        public UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
37        ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
38        ↳ links, header) { }
39
40        /// <summary>
41        /// <para>
42        /// Gets the left reference using the specified node.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="node">
47        /// <para>The node.</para>
48        /// <para></para>
49        /// </param>
50        /// <returns>
51        /// <para>The ref ulong</para>
52        /// <para></para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        protected override ref ulong GetLeftReference(ulong node) => ref
56        ↳ Links[node].LeftAsSource;
57
58        /// <summary>
59        /// <para>
60        /// Gets the right reference using the specified node.
61        /// </para>
62        /// <para></para>
63        /// </summary>
64        /// <param name="node">
65        /// <para>The node.</para>
66        /// <para></para>
67        /// </param>
68        /// <returns>
69        /// <para>The ref ulong</para>
70        /// <para></para>
71        /// </returns>
72        [MethodImpl(MethodImplOptions.AggressiveInlining)]
73        protected override ref ulong GetRightReference(ulong node) => ref
74        ↳ Links[node].RightAsSource;
```



```

72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
        ↳ left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
        ↳ right;
137
138    /// <summary>
139    /// <para>
140    /// Gets the size using the specified node.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <param name="node">
145    /// <para>The node.</para>
146    /// <para></para>
147    /// </param>

```

```

148     /// <returns>
149     /// <para>The ulong</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="node">
162     /// <para>The node.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsSource =
171         ↪ size;
172
173     /// <summary>
174     /// <para>
175     /// Gets the tree root.
176     /// </para>
177     /// <para></para>
178     /// </summary>
179     /// <returns>
180     /// <para>The ulong</para>
181     /// <para></para>
182     /// </returns>
183     [MethodImpl(MethodImplOptions.AggressiveInlining)]
184     protected override ulong GetTreeRoot() => Header->RootAsSource;
185
186     /// <summary>
187     /// <para>
188     /// Gets the base part value using the specified link.
189     /// </para>
190     /// <para></para>
191     /// </summary>
192     /// <param name="link">
193     /// <para>The link.</para>
194     /// <para></para>
195     /// </param>
196     /// <returns>
197     /// <para>The ulong</para>
198     /// <para></para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override ulong GetBasePartValue(ulong link) => Links[link].Source;
202
203     /// <summary>
204     /// <para>
205     /// Determines whether this instance first is to the left of second.
206     /// </para>
207     /// <para></para>
208     /// </summary>
209     /// <param name="firstSource">
210     /// <para>The first source.</para>
211     /// <para></para>
212     /// </param>
213     /// <param name="firstTarget">
214     /// <para>The first target.</para>
215     /// <para></para>
216     /// </param>
217     /// <param name="secondSource">
218     /// <para>The second source.</para>
219     /// <para></para>
220     /// </param>
221     /// <param name="secondTarget">
222     /// <para>The second target.</para>
223     /// <para></para>
224     /// </param>
225     /// <returns>

```

```

225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
230         ↪ ulong secondSource, ulong secondTarget)
231         => firstSource < secondSource || (firstSource == secondSource && firstTarget <
232             ↪ secondTarget);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondTarget">
253     /// <para>The second target.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
262         ↪ ulong secondSource, ulong secondTarget)
263         => firstSource > secondSource || (firstSource == secondSource && firstTarget >
264             ↪ secondTarget);
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>
270     /// <para></para>
271     /// </summary>
272     /// <param name="node">
273     /// <para>The node.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(ulong node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsSource = OUL;
281         link.RightAsSource = OUL;
282         link.SizeAsSource = OUL;
283     }
284 }

```

1.106 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.c

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links sources size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksSourcesSizeBalancedTreeMethods :
15         ↪ UInt64LinksSizeBalancedTreeMethodsBase

```

```

15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
35         ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
36         ↳ { }
37
38     /// <summary>
39     /// <para>
40     /// Gets the left reference using the specified node.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     /// <param name="node">
45     /// <para>The node.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The ref ulong</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override ref ulong GetLeftReference(ulong node) => ref
54         ↳ Links[node].LeftAsSource;
55
56     /// <summary>
57     /// <para>
58     /// Gets the right reference using the specified node.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="node">
63     /// <para>The node.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The ref ulong</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref ulong GetRightReference(ulong node) => ref
72         ↳ Links[node].RightAsSource;
73
74     /// <summary>
75     /// <para>
76     /// Gets the left using the specified node.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="node">
81     /// <para>The node.</para>
82     /// <para></para>
83     /// </param>
84     /// <returns>
85     /// <para>The ulong</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
90
91     /// <summary>
92     /// <para>

```

```

89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
    ↪ left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
    ↪ right;
137
138    /// <summary>
139    /// <para>
140    /// Gets the size using the specified node.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <param name="node">
145    /// <para>The node.</para>
146    /// <para></para>
147    /// </param>
148    /// <returns>
149    /// <para>The ulong</para>
150    /// <para></para>
151    /// </returns>
152    [MethodImpl(MethodImplOptions.AggressiveInlining)]
153    protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;
154
155    /// <summary>
156    /// <para>
157    /// Sets the size using the specified node.
158    /// </para>
159    /// <para></para>
160    /// </summary>
161    /// <param name="node">
162    /// <para>The node.</para>
163    /// <para></para>
164    /// </param>

```

```

165     /// <param name="size">
166     /// <para>The size.</para>
167     /// </para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsSource =
        ↳ size;

171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// </para>
177     /// </summary>
178     /// <returns>
179     /// <para>The ulong</para>
180     /// </para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override ulong GetTreeRoot() => Header->RootAsSource;

184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified link.
188     /// </para>
189     /// </para>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>
193     /// </para>
194     /// </param>
195     /// <returns>
196     /// <para>The ulong</para>
197     /// </para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override ulong GetBasePartValue(ulong link) => Links[link].Source;

201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// </para>
207     /// </summary>
208     /// <param name="firstSource">
209     /// <para>The first source.</para>
210     /// </para>
211     /// </param>
212     /// <param name="firstTarget">
213     /// <para>The first target.</para>
214     /// </para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// </para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// </para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// </para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
        ↳ ulong secondSource, ulong secondTarget)
230     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
        ↳ secondTarget);

231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// </para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>

```

```

240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
        ↪     ulong secondSource, ulong secondTarget)
260         => firstSource > secondSource || (firstSource == secondSource && firstTarget >
        ↪     secondTarget);
261
262     /// <summary>
263     /// <para>
264     /// Clears the node using the specified node.
265     /// </para>
266     /// <para></para>
267     /// </summary>
268     /// <param name="node">
269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(ulong node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsSource = OUL;
277         link.RightAsSource = OUL;
278         link.SizeAsSource = OUL;
279     }
280 }
281 }

```

1.107 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
        ↪     UInt64LinksAvlBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>

```

```

32     /// <para></para>
33     /// </param>
34     public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
    ↪     RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
    ↪     { }
35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪     Links[node].LeftAsTarget;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↪     Links[node].RightAsTarget;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>

```



```

106     /// Sets the left using the specified node.
107     /// </para>
108     /// <para></para>
109     /// </summary>
110     /// <param name="node">
111     /// <para>The node.</para>
112     /// <para></para>
113     /// </param>
114     /// <param name="left">
115     /// <para>The left.</para>
116     /// <para></para>
117     /// </param>
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]
119     protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
        ↳ left;
120
121     /// <summary>
122     /// <para>
123     /// Sets the right using the specified node.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="node">
128     /// <para>The node.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
        ↳ right;
137
138     /// <summary>
139     /// <para>
140     /// Gets the size using the specified node.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="node">
145     /// <para>The node.</para>
146     /// <para></para>
147     /// </param>
148     /// <returns>
149     /// <para>The ulong</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsTarget);
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="node">
162     /// <para>The node.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
        ↳ Links[node].SizeAsTarget, size);
171
172     /// <summary>
173     /// <para>
174     /// Determines whether this instance get left is child.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <param name="node">
179     /// <para>The node.</para>
180     /// <para></para>

```

```

181     /// </param>
182     /// <returns>
183     /// <para>The bool</para>
184     /// <para></para>
185     /// </returns>
186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
187     protected override bool GetLeftIsChild(ulong node) =>
188         ↪ GetLeftIsChildValue(Links[node].SizeAsTarget);
189
189     /// <summary>
190     /// <para>
191     /// Sets the left is child using the specified node.
192     /// </para>
193     /// <para></para>
194     /// </summary>
195     /// <param name="node">
196     /// <para>The node.</para>
197     /// <para></para>
198     /// </param>
199     /// <param name="value">
200     /// <para>The value.</para>
201     /// <para></para>
202     /// </param>
203     [MethodImpl(MethodImplOptions.AggressiveInlining)]
204     protected override void SetLeftIsChild(ulong node, bool value) =>
205         ↪ SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);
206
206     /// <summary>
207     /// <para>
208     /// Determines whether this instance get right is child.
209     /// </para>
210     /// <para></para>
211     /// </summary>
212     /// <param name="node">
213     /// <para>The node.</para>
214     /// <para></para>
215     /// </param>
216     /// <returns>
217     /// <para>The bool</para>
218     /// <para></para>
219     /// </returns>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override bool GetRightIsChild(ulong node) =>
222         ↪ GetRightIsChildValue(Links[node].SizeAsTarget);
223
223     /// <summary>
224     /// <para>
225     /// Sets the right is child using the specified node.
226     /// </para>
227     /// <para></para>
228     /// </summary>
229     /// <param name="node">
230     /// <para>The node.</para>
231     /// <para></para>
232     /// </param>
233     /// <param name="value">
234     /// <para>The value.</para>
235     /// <para></para>
236     /// </param>
237     [MethodImpl(MethodImplOptions.AggressiveInlining)]
238     protected override void SetRightIsChild(ulong node, bool value) =>
239         ↪ SetRightIsChildValue(ref Links[node].SizeAsTarget, value);
240
240     /// <summary>
241     /// <para>
242     /// Gets the balance using the specified node.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="node">
247     /// <para>The node.</para>
248     /// <para></para>
249     /// </param>
250     /// <returns>
251     /// <para>The sbyte</para>
252     /// <para></para>
253     /// </returns>
254     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

255     protected override sbyte GetBalance(ulong node) =>
256         ↳ GetBalanceValue(Links[node].SizeAsTarget);
257
258     /// <summary>
259     /// <para>
260     /// Sets the balance using the specified node.
261     /// </para>
262     /// </summary>
263     /// <param name="node">
264     /// <para>The node.</para>
265     /// </param>
266     /// <param name="value">
267     /// <para>The value.</para>
268     /// </param>
269     [MethodImpl(MethodImplOptions.AggressiveInlining)]
270     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
271         ↳ Links[node].SizeAsTarget, value);
272
273     /// <summary>
274     /// <para>
275     /// Gets the tree root.
276     /// </para>
277     /// </summary>
278     /// <returns>
279     /// <para>The ulong</para>
280     /// </returns>
281     [MethodImpl(MethodImplOptions.AggressiveInlining)]
282     protected override ulong GetTreeRoot() => Header->RootAsTarget;
283
284     /// <summary>
285     /// <para>
286     /// Gets the base part value using the specified link.
287     /// </para>
288     /// </summary>
289     /// <param name="link">
290     /// <para>The link.</para>
291     /// </param>
292     /// <returns>
293     /// <para>The ulong</para>
294     /// </returns>
295     [MethodImpl(MethodImplOptions.AggressiveInlining)]
296     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
297
298     /// <summary>
299     /// <para>
300     /// Determines whether this instance first is to the left of second.
301     /// </para>
302     /// </summary>
303     /// <param name="firstSource">
304     /// <para>The first source.</para>
305     /// </param>
306     /// <param name="firstTarget">
307     /// <para>The first target.</para>
308     /// </param>
309     /// <param name="secondSource">
310     /// <para>The second source.</para>
311     /// </param>
312     /// <param name="secondTarget">
313     /// <para>The second target.</para>
314     /// </param>
315     /// <returns>
316     /// <para>The bool</para>
317     /// </returns>
318     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

331     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
332         ↪ ulong secondSource, ulong secondTarget)
333         => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
334             ↪ secondSource);
335
336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the right of second.
339     /// </para>
340     /// </summary>
341     /// <param name="firstSource">
342     /// <para>The first source.</para>
343     /// </param>
344     /// <param name="firstTarget">
345     /// <para>The first target.</para>
346     /// </param>
347     /// <param name="secondSource">
348     /// <para>The second source.</para>
349     /// </param>
350     /// <param name="secondTarget">
351     /// <para>The second target.</para>
352     /// </param>
353     /// <returns>
354     /// <para>The bool</para>
355     /// </returns>
356     [MethodImpl(MethodImplOptions.AggressiveInlining)]
357     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
358         ↪ ulong secondSource, ulong secondTarget)
359         => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
360             ↪ secondSource);
361
362     /// <summary>
363     /// <para>
364     /// Clears the node using the specified node.
365     /// </para>
366     /// </summary>
367     /// <param name="node">
368     /// <para>The node.</para>
369     /// </param>
370     [MethodImpl(MethodImplOptions.AggressiveInlining)]
371     protected override void ClearNode(ulong node)
372     {
373         ref var link = ref Links[node];
374         link.LeftAsTarget = OUL;
375         link.RightAsTarget = OUL;
376         link.SizeAsTarget = OUL;
377     }
378 }
379
380 }
381
382 }
383

```

1.108 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets recursionless size balanced tree methods.
10     /// </para>
11     /// </summary>
12     /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
13     public unsafe class UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods :
14         ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>

```

```

18     /// Initializes a new <see
    ↪ cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
    ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
    ↪ links, header) { }

35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ Links[node].LeftAsTarget;

52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↪ Links[node].RightAsTarget;

69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.

```

```

90    /// </para>
91    /// <para></para>
92    /// </summary>
93    /// <param name="node">
94    /// <para>The node.</para>
95    /// <para></para>
96    /// </param>
97    /// <returns>
98    /// <para>The ulong</para>
99    /// <para></para>
100   /// </returns>
101   [MethodImpl(MethodImplOptions.AggressiveInlining)]
102   protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
103
104   /// <summary>
105   /// <para>
106   /// Sets the left using the specified node.
107   /// </para>
108   /// <para></para>
109   /// </summary>
110   /// <param name="node">
111   /// <para>The node.</para>
112   /// <para></para>
113   /// </param>
114   /// <param name="left">
115   /// <para>The left.</para>
116   /// <para></para>
117   /// </param>
118   [MethodImpl(MethodImplOptions.AggressiveInlining)]
119   protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
120   ↪ left;
121
122   /// <summary>
123   /// <para>
124   /// Sets the right using the specified node.
125   /// </para>
126   /// <para></para>
127   /// </summary>
128   /// <param name="node">
129   /// <para>The node.</para>
130   /// <para></para>
131   /// </param>
132   /// <param name="right">
133   /// <para>The right.</para>
134   /// <para></para>
135   /// </param>
136   [MethodImpl(MethodImplOptions.AggressiveInlining)]
137   protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
138   ↪ right;
139
140   /// <summary>
141   /// <para>
142   /// Gets the size using the specified node.
143   /// </para>
144   /// <para></para>
145   /// </summary>
146   /// <param name="node">
147   /// <para>The node.</para>
148   /// <para></para>
149   /// </param>
150   /// <returns>
151   /// <para>The ulong</para>
152   /// <para></para>
153   /// </returns>
154   [MethodImpl(MethodImplOptions.AggressiveInlining)]
155   protected override ulong GetSize(ulong node) => Links[node].SizeAsTarget;
156
157   /// <summary>
158   /// <para>
159   /// Sets the size using the specified node.
160   /// </para>
161   /// <para></para>
162   /// </summary>
163   /// <param name="node">
164   /// <para>The node.</para>
165   /// <para></para>
166   /// </param>
167   /// <param name="size">

```

```

166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsTarget =
        ↳ size;

171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <returns>
179     /// <para>The ulong</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override ulong GetTreeRoot() => Header->RootAsTarget;
184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified link.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The ulong</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="firstSource">
209     /// <para>The first source.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="firstTarget">
213     /// <para>The first target.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
        ↳ ulong secondSource, ulong secondTarget)
230         => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
        ↳ secondSource);

231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>

```

[illegible]

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links targets size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksTargetsSizeBalancedTreeMethods :
15        ↪ UInt64LinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksTargetsSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// </param>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="header">
31        /// <para>A header.</para>
32        /// <para></para>

```



```

33     /// </param>
34 public UInt64LinksTargetsSizeBalancedTreeMethods(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
    ↳ { }
35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↳ Links[node].LeftAsTarget;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↳ Links[node].RightAsTarget;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.

```

```

107     /// </para>
108     /// <para></para>
109     /// </summary>
110     /// <param name="node">
111     /// <para>The node.</para>
112     /// <para></para>
113     /// </param>
114     /// <param name="left">
115     /// <para>The left.</para>
116     /// <para></para>
117     /// </param>
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]
119     protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
        ↳ left;
120
121     /// <summary>
122     /// <para>
123     /// Sets the right using the specified node.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="node">
128     /// <para>The node.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
        ↳ right;
137
138     /// <summary>
139     /// <para>
140     /// Gets the size using the specified node.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="node">
145     /// <para>The node.</para>
146     /// <para></para>
147     /// </param>
148     /// <returns>
149     /// <para>The ulong</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override ulong GetSize(ulong node) => Links[node].SizeAsTarget;
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="node">
162     /// <para>The node.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsTarget =
        ↳ size;
171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <returns>
179     /// <para>The ulong</para>
180     /// <para></para>
181     /// </returns>

```

```

182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 protected override ulong GetTreeRoot() => Header->RootAsTarget;
184
185 /// <summary>
186 /// <para>
187 /// Gets the base part value using the specified link.
188 /// </para>
189 /// <para></para>
190 /// </summary>
191 /// <param name="link">
192 /// <para>The link.</para>
193 /// <para></para>
194 /// </param>
195 /// <returns>
196 /// <para>The ulong</para>
197 /// <para></para>
198 /// </returns>
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
201
202 /// <summary>
203 /// <para>
204 /// Determines whether this instance first is to the left of second.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <param name="firstSource">
209 /// <para>The first source.</para>
210 /// <para></para>
211 /// </param>
212 /// <param name="firstTarget">
213 /// <para>The first target.</para>
214 /// <para></para>
215 /// </param>
216 /// <param name="secondSource">
217 /// <para>The second source.</para>
218 /// <para></para>
219 /// </param>
220 /// <param name="secondTarget">
221 /// <para>The second target.</para>
222 /// <para></para>
223 /// </param>
224 /// <returns>
225 /// <para>The bool</para>
226 /// <para></para>
227 /// </returns>
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
230     ↪ ulong secondSource, ulong secondTarget)
231     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
232     ↪ secondSource);
233
234 /// <summary>
235 /// <para>
236 /// Determines whether this instance first is to the right of second.
237 /// </para>
238 /// <para></para>
239 /// </summary>
240 /// <param name="firstSource">
241 /// <para>The first source.</para>
242 /// <para></para>
243 /// </param>
244 /// <param name="firstTarget">
245 /// <para>The first target.</para>
246 /// <para></para>
247 /// </param>
248 /// <param name="secondSource">
249 /// <para>The second source.</para>
250 /// <para></para>
251 /// </param>
252 /// <param name="secondTarget">
253 /// <para>The second target.</para>
254 /// <para></para>
255 /// </param>
256 /// <returns>
257 /// <para>The bool</para>
258 /// <para></para>
259 /// </returns>

```

```

258 [MethodImpl(MethodImplOptions.AggressiveInlining)]
259 protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
    ↳ ulong secondSource, ulong secondTarget)
260 => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
    ↳ secondSource);
261
262 /// <summary>
263 /// <para>
264 /// Clears the node using the specified node.
265 /// </para>
266 /// <para></para>
267 /// </summary>
268 /// <param name="node">
269 /// <para>The node.</para>
270 /// </param>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 protected override void ClearNode(ulong node)
273 {
274     ref var link = ref Links[node];
275     link.LeftAsTarget = OUL;
276     link.RightAsTarget = OUL;
277     link.SizeAsTarget = OUL;
278 }
279 }
280 }
281 }

```

1.110 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Memory;
4 using Platform.Singletons;
5 using Platform.Data.Doublets.Memory.United.Generic;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ↳ organizing the storage of links with addresses represented as <see cref="ulong">
14     ↳ </para>
15     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
16     ↳ размером, для организации хранения связей с адресами представленными в виде <see
17     ↳ cref="ulong"></para>
18     /// </summary>
19     public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
20     {
21         private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
23         private LinksHeader<ulong>* _header;
24         private RawLink<ulong>* _links;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="address">
33         /// <para>A address.</para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
37
38         /// <summary>
39         /// Создает экземпляр базы данных Links в файле по указанному адресу, с указанным
40         ↳ минимальным шагом расширения базы данных.
41         /// </summary>
42         /// <param name="address">Полный путь к файлу базы данных.</param>
43         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
44         ↳ байтах.</param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         ↳ FileMappedResizableDirectMemory(address, memoryReservationStep),
48         ↳ memoryReservationStep) { }
49
50         /// <summary>

```

```

44     /// <para>
45     /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
46     /// </para>
47     /// </summary>
48     /// <param name="memory">
49     /// <para>A memory.</para>
50     /// </param>
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
53     ↪ DefaultLinksSizeStep) { }
54
55     /// <summary>
56     /// <para>
57     /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
58     /// </para>
59     /// </summary>
60     /// <param name="memory">
61     /// <para>A memory.</para>
62     /// </param>
63     /// <param name="memoryReservationStep">
64     /// <para>A memory reservation step.</para>
65     /// </param>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
68     ↪ memoryReservationStep) : this(memory, memoryReservationStep,
69     ↪ Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }
70
71     /// <summary>
72     /// <para>
73     /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
74     /// </para>
75     /// </summary>
76     /// <param name="memory">
77     /// <para>A memory.</para>
78     /// </param>
79     /// <param name="memoryReservationStep">
80     /// <para>A memory reservation step.</para>
81     /// </param>
82     /// <param name="constants">
83     /// <para>A constants.</para>
84     /// </param>
85     /// <param name="indexTreeType">
86     /// <para>A index tree type.</para>
87     /// </param>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
90     ↪ memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
91     ↪ : base(memory, memoryReservationStep, constants)
92     {
93         if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
94         {
95             _createSourceTreeMethods = () => new
96             ↪ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
97             _createTargetTreeMethods = () => new
98             ↪ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
99         }
100         else if (indexTreeType == IndexTreeType.SizeBalancedTree)
101         {
102             _createSourceTreeMethods = () => new
103             ↪ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
104             _createTargetTreeMethods = () => new
105             ↪ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
106         }
107         else
108         {
109

```

```

110         _createSourceTreeMethods = () => new
111         ↪ UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
112         ↪ _header);
113     _createTargetTreeMethods = () => new
114     ↪ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
115     ↪ _header);
116 }
117 Init(memory, memoryReservationStep);
118 }
119
120 /// <summary>
121 /// <para>
122 /// Sets the pointers using the specified memory.
123 /// </para>
124 /// <para></para>
125 /// </summary>
126 /// <param name="memory">
127 /// <para>The memory.</para>
128 /// <para></para>
129 /// </param>
130 [MethodImpl(MethodImplOptions.AggressiveInlining)]
131 protected override void SetPointers(IResizableDirectMemory memory)
132 {
133     _header = (LinksHeader<ulong>*)memory.Pointer;
134     _links = (RawLink<ulong>*)memory.Pointer;
135     SourcesTreeMethods = _createSourceTreeMethods();
136     TargetsTreeMethods = _createTargetTreeMethods();
137     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
138 }
139
140 /// <summary>
141 /// <para>
142 /// Resets the pointers.
143 /// </para>
144 /// <para></para>
145 /// </summary>
146 [MethodImpl(MethodImplOptions.AggressiveInlining)]
147 protected override void ResetPointers()
148 {
149     base.ResetPointers();
150     _links = null;
151     _header = null;
152 }
153
154 /// <summary>
155 /// <para>
156 /// Gets the header reference.
157 /// </para>
158 /// <para></para>
159 /// </summary>
160 /// <returns>
161 /// <para>A ref links header of ulong</para>
162 /// <para></para>
163 /// </returns>
164 [MethodImpl(MethodImplOptions.AggressiveInlining)]
165 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
166
167 /// <summary>
168 /// <para>
169 /// Gets the link reference using the specified link index.
170 /// </para>
171 /// <para></para>
172 /// </summary>
173 /// <param name="linkIndex">
174 /// <para>The link index.</para>
175 /// <para></para>
176 /// </param>
177 /// <returns>
178 /// <para>A ref raw link of ulong</para>
179 /// <para></para>
180 /// </returns>
181 [MethodImpl(MethodImplOptions.AggressiveInlining)]
182 protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
    ↪ _links[linkIndex];

```

```

183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <param name="first">
187     /// <para>The first.</para>
188     /// <para></para>
189     /// </param>
190     /// <param name="second">
191     /// <para>The second.</para>
192     /// <para></para>
193     /// </param>
194     /// <returns>
195     /// <para>The bool</para>
196     /// <para></para>
197     /// </returns>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override bool AreEqual(ulong first, ulong second) => first == second;
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance less than.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="first">
208     /// <para>The first.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="second">
212     /// <para>The second.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>The bool</para>
217     /// <para></para>
218     /// </returns>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override bool LessThan(ulong first, ulong second) => first < second;
221
222     /// <summary>
223     /// <para>
224     /// Determines whether this instance less or equal than.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="first">
229     /// <para>The first.</para>
230     /// <para></para>
231     /// </param>
232     /// <param name="second">
233     /// <para>The second.</para>
234     /// <para></para>
235     /// </param>
236     /// <returns>
237     /// <para>The bool</para>
238     /// <para></para>
239     /// </returns>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
242
243     /// <summary>
244     /// <para>
245     /// Determines whether this instance greater than.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="first">
250     /// <para>The first.</para>
251     /// <para></para>
252     /// </param>
253     /// <param name="second">
254     /// <para>The second.</para>
255     /// <para></para>
256     /// </param>
257     /// <returns>
258     /// <para>The bool</para>
259     /// <para></para>
260     /// </returns>

```

```

261 [MethodImpl(MethodImplOptions.AggressiveInlining)]
262 protected override bool GreaterThan(ulong first, ulong second) => first > second;
263
264 /// <summary>
265 /// <para>
266 /// Determines whether this instance greater or equal than.
267 /// </para>
268 /// <para></para>
269 /// </summary>
270 /// <param name="first">
271 /// <para>The first.</para>
272 /// <para></para>
273 /// </param>
274 /// <param name="second">
275 /// <para>The second.</para>
276 /// <para></para>
277 /// </param>
278 /// <returns>
279 /// <para>The bool</para>
280 /// <para></para>
281 /// </returns>
282 [MethodImpl(MethodImplOptions.AggressiveInlining)]
283 protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
284
285 /// <summary>
286 /// <para>
287 /// Gets the zero.
288 /// </para>
289 /// <para></para>
290 /// </summary>
291 /// <returns>
292 /// <para>The ulong</para>
293 /// <para></para>
294 /// </returns>
295 [MethodImpl(MethodImplOptions.AggressiveInlining)]
296 protected override ulong GetZero() => 0UL;
297
298 /// <summary>
299 /// <para>
300 /// Gets the one.
301 /// </para>
302 /// <para></para>
303 /// </summary>
304 /// <returns>
305 /// <para>The ulong</para>
306 /// <para></para>
307 /// </returns>
308 [MethodImpl(MethodImplOptions.AggressiveInlining)]
309 protected override ulong GetOne() => 1UL;
310
311 /// <summary>
312 /// <para>
313 /// Converts the to int 64 using the specified value.
314 /// </para>
315 /// <para></para>
316 /// </summary>
317 /// <param name="value">
318 /// <para>The value.</para>
319 /// <para></para>
320 /// </param>
321 /// <returns>
322 /// <para>The long</para>
323 /// <para></para>
324 /// </returns>
325 [MethodImpl(MethodImplOptions.AggressiveInlining)]
326 protected override long ConvertToInt64(ulong value) => (long)value;
327
328 /// <summary>
329 /// <para>
330 /// Converts the to address using the specified value.
331 /// </para>
332 /// <para></para>
333 /// </summary>
334 /// <param name="value">
335 /// <para>The value.</para>
336 /// <para></para>
337 /// </param>
338 /// <returns>

```



```

339    /// <para>The ulong</para>
340    /// <para></para>
341    /// </returns>
342    [MethodImpl(MethodImplOptions.AggressiveInlining)]
343    protected override ulong ConvertToAddress(long value) => (ulong)value;
344
345    /// <summary>
346    /// <para>
347    /// Adds the first.
348    /// </para>
349    /// <para></para>
350    /// </summary>
351    /// <param name="first">
352    /// <para>The first.</para>
353    /// <para></para>
354    /// </param>
355    /// <param name="second">
356    /// <para>The second.</para>
357    /// <para></para>
358    /// </param>
359    /// <returns>
360    /// <para>The ulong</para>
361    /// <para></para>
362    /// </returns>
363    [MethodImpl(MethodImplOptions.AggressiveInlining)]
364    protected override ulong Add(ulong first, ulong second) => first + second;
365
366    /// <summary>
367    /// <para>
368    /// Subtracts the first.
369    /// </para>
370    /// <para></para>
371    /// </summary>
372    /// <param name="first">
373    /// <para>The first.</para>
374    /// <para></para>
375    /// </param>
376    /// <param name="second">
377    /// <para>The second.</para>
378    /// <para></para>
379    /// </param>
380    /// <returns>
381    /// <para>The ulong</para>
382    /// <para></para>
383    /// </returns>
384    [MethodImpl(MethodImplOptions.AggressiveInlining)]
385    protected override ulong Subtract(ulong first, ulong second) => first - second;
386
387    /// <summary>
388    /// <para>
389    /// Increments the link.
390    /// </para>
391    /// <para></para>
392    /// </summary>
393    /// <param name="link">
394    /// <para>The link.</para>
395    /// <para></para>
396    /// </param>
397    /// <returns>
398    /// <para>The ulong</para>
399    /// <para></para>
400    /// </returns>
401    [MethodImpl(MethodImplOptions.AggressiveInlining)]
402    protected override ulong Increment(ulong link) => ++link;
403
404    /// <summary>
405    /// <para>
406    /// Decrements the link.
407    /// </para>
408    /// <para></para>
409    /// </summary>
410    /// <param name="link">
411    /// <para>The link.</para>
412    /// <para></para>
413    /// </param>
414    /// <returns>
415    /// <para>The ulong</para>
416    /// <para></para>

```

```

417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override ulong Decrement(ulong link) => --link;
420 }
421 }

```

1.111 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 unused links list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UnusedLinksListMethods{ulong}"/>
15     public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<ulong>
16     {
17         private readonly RawLink<ulong>* _links;
18         private readonly LinksHeader<ulong>* _header;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public UInt64UnusedLinksListMethods(RawLink<ulong>* links, LinksHeader<ulong>* header)
36             : base((byte*)links, (byte*)header)
37         {
38             _links = links;
39             _header = header;
40         }
41
42         /// <summary>
43         /// <para>
44         /// Gets the link reference using the specified link.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="link">
49         /// <para>The link.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>A ref raw link of ulong</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref _links[link];
58
59         /// <summary>
60         /// <para>
61         /// Gets the header reference.
62         /// </para>
63         /// <para></para>
64         /// </summary>
65         /// <returns>
66         /// <para>A ref links header of ulong</para>
67         /// <para></para>
68         /// </returns>
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
71     }
72 }

```

1.112 ./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the properties operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}" />
16     /// <seealso cref="IProperties{TLink, TLink, TLink}" />
17     public class PropertiesOperator<TLink> : LinksOperatorBase<TLink>, IProperties<TLink, TLink,
18     ↪ TLink>
19     {
20         private static readonly EqualityComparer<TLink> _equalityComparer =
21         ↪ EqualityComparer<TLink>.Default;
22
23         /// <summary>
24         /// <para>
25         /// Initializes a new <see cref="PropertiesOperator" /> instance.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         /// <param name="links">
30         /// <para>A links.</para>
31         /// <para></para>
32         /// </param>
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public PropertiesOperator(ILinks<TLink> links) : base(links) { }
35
36         /// <summary>
37         /// <para>
38         /// Gets the value using the specified object.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="@object">
43         /// <para>The object.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="property">
47         /// <para>The property.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The link</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public TLink GetValue(TLink @object, TLink property)
56         {
57             var links = _links;
58             var objectProperty = links.SearchOrDefault(@object, property);
59             if (_equalityComparer.Equals(objectProperty, default))
60             {
61                 return default;
62             }
63             var constants = links.Constants;
64             var any = constants.Any;
65             var query = new Link<TLink>(any, objectProperty, any);
66             var valueLink = links.SingleOrDefault(query);
67             if (valueLink == null)
68             {
69                 return default;
70             }
71             return links.GetTarget(valueLink[constants.IndexPart]);
72         }
73
74         /// <summary>
75         /// <para>
76         /// Sets the value using the specified object.
77         /// </para>
78         /// <para></para>

```

```

77     /// </summary>
78     /// <param name="@object">
79     /// <para>The object.</para>
80     /// <para></para>
81     /// </param>
82     /// <param name="property">
83     /// <para>The property.</para>
84     /// <para></para>
85     /// </param>
86     /// <param name="value">
87     /// <para>The value.</para>
88     /// <para></para>
89     /// </param>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public void SetValue(TLink @object, TLink property, TLink value)
92     {
93         var links = _links;
94         var objectProperty = links.GetOrCreate(@object, property);
95         links.DeleteMany(links.AllIndices(links.Constants.Any, objectProperty));
96         links.GetOrCreate(objectProperty, value);
97     }
98 }
99 }

```

1.113 ./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the property operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}">
16     /// <seealso cref="IProperty{TLink, TLink}">
17     public class PropertyOperator<TLink> : LinksOperatorBase<TLink>, IProperty<TLink, TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
20             EqualityComparer<TLink>.Default;
21         private readonly TLink _propertyMarker;
22         private readonly TLink _propertyValueMarker;
23
24         /// <summary>
25         /// <para>
26         /// Initializes a new <see cref="PropertyOperator"/> instance.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         /// <param name="links">
31         /// <para>A links.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="propertyMarker">
35         /// <para>A property marker.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="propertyValueMarker">
39         /// <para>A property value marker.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
44             propertyValueMarker) : base(links)
45         {
46             _propertyMarker = propertyMarker;
47             _propertyValueMarker = propertyValueMarker;
48         }
49
50         /// <summary>
51         /// <para>
52         /// Gets the link.
53         /// </para>
54         /// <para></para>
55     }

```

```

53     /// </summary>
54     /// <param name="link">
55     /// <para>The link.</para>
56     /// <para></para>
57     /// </param>
58     /// <returns>
59     /// <para>The link</para>
60     /// <para></para>
61     /// </returns>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public TLink Get(TLink link)
64     {
65         var property = _links.SearchOrDefault(link, _propertyMarker);
66         return GetValue(GetContainer(property));
67     }
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     private TLink GetContainer(TLink property)
70     {
71         var valueContainer = default(TLink);
72         if (_equalityComparer.Equals(property, default))
73         {
74             return valueContainer;
75         }
76         var links = _links;
77         var constants = links.Constants;
78         var countinueConstant = constants.Continue;
79         var breakConstant = constants.Break;
80         var anyConstant = constants.Any;
81         var query = new Link<TLink>(anyConstant, property, anyConstant);
82         links.Each(candidate =>
83         {
84             var candidateTarget = links.GetTarget(candidate);
85             var valueTarget = links.GetTarget(candidateTarget);
86             if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
87             {
88                 valueContainer = links.GetIndex(candidate);
89                 return breakConstant;
90             }
91             return countinueConstant;
92         }, query);
93         return valueContainer;
94     }
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
97     ↪ ? default : _links.GetTarget(container);
98     /// <summary>
99     /// <para>
100     /// Sets the link.
101     /// </para>
102     /// <para></para>
103     /// </summary>
104     /// <param name="link">
105     /// <para>The link.</para>
106     /// <para></para>
107     /// </param>
108     /// <param name="value">
109     /// <para>The value.</para>
110     /// <para></para>
111     /// </param>
112     [MethodImpl(MethodImplOptions.AggressiveInlining)]
113     public void Set(TLink link, TLink value)
114     {
115         var links = _links;
116         var property = links.GetOrCreate(link, _propertyMarker);
117         var container = GetContainer(property);
118         if (_equalityComparer.Equals(container, default))
119         {
120             links.GetOrCreate(property, value);
121         }
122         else
123         {
124             links.Update(container, property, value);
125         }
126     }
127 }
128 }

```

1.114 ./csharp/Platform.Data.Doublets/Stacks/Stack.cs

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections.Stacks;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Stacks
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the stack.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}">
17     /// <seealso cref="IStack{TLink}">
18     public class Stack<TLink> : LinksOperatorBase<TLink>, IStack<TLink>
19     {
20         private static readonly EqualityComparer<TLink> _equalityComparer =
21             ↪ EqualityComparer<TLink>.Default;
22         private readonly TLink _stack;
23
24         /// <summary>
25         /// <para>
26         /// Gets the is empty value.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         public bool IsEmpty
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get => _equalityComparer.Equals(Peek(), _stack);
34         }
35
36         /// <summary>
37         /// <para>
38         /// Initializes a new <see cref="Stack" /> instance.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="links">
43         /// <para>A links.</para>
44         /// </param>
45         /// <param name="stack">
46         /// <para>A stack.</para>
47         /// </param>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public Stack(ILinks<TLink> links, TLink stack) : base(links) => _stack = stack;
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         private TLink GetStackMarker() => _links.GetSource(_stack);
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         private TLink GetTop() => _links.GetTarget(_stack);
54
55         /// <summary>
56         /// <para>
57         /// Peeks this instance.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <returns>
62         /// <para>The link</para>
63         /// <para></para>
64         /// </returns>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public TLink Peek() => _links.GetTarget(GetTop());
67
68         /// <summary>
69         /// <para>
70         /// Pops this instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <returns>
75         /// <para>The element.</para>
76         /// <para></para>
77         /// </returns>

```

```

78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public TLink Pop()
81     {
82         var element = Peek();
83         if (!_equalityComparer.Equals(element, _stack))
84         {
85             var top = GetTop();
86             var previousTop = _links.GetSource(top);
87             _links.Update(_stack, GetStackMarker(), previousTop);
88             _links.Delete(top);
89         }
90         return element;
91     }
92
93     /// <summary>
94     /// <para>
95     /// Pushes the element.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="element">
100    /// <para>The element.</para>
101    /// <para></para>
102    /// </param>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    public void Push(TLink element) => _links.Update(_stack, GetStackMarker(),
105        ↪ _links.GetOrCreate(GetTop(), element));
106 }

```

1.115 ./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Stacks
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the stack extensions.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public static class StackExtensions
14     {
15         /// <summary>
16         /// <para>
17         /// Creates the stack using the specified links.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         /// <typeparam name="TLink">
22         /// <para>The link.</para>
23         /// <para></para>
24         /// </typeparam>
25         /// <param name="links">
26         /// <para>The links.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="stackMarker">
30         /// <para>The stack marker.</para>
31         /// <para></para>
32         /// </param>
33         /// <returns>
34         /// <para>The stack.</para>
35         /// <para></para>
36         /// </returns>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public static TLink CreateStack<TLink>(this ILinks<TLink> links, TLink stackMarker)
39         {
40             var stackPoint = links.CreatePoint();
41             var stack = links.Update(stackPoint, stackMarker, stackPoint);
42             return stack;
43         }
44     }
45 }

```

1.116 ./csharp/Platform.Data.Doublets/SynchronizedLinks.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Data.Doublets;
5  using Platform.Delegates;
6  using Platform.Threading.Synchronization;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets
11 {
12     /// <remarks>
13     /// TODO: Autogeneration of synchronized wrapper (decorator).
14     /// TODO: Try to unfold code of each method using IL generation for performance improvements.
15     /// TODO: Or even to unfold multiple layers of implementations.
16     /// </remarks>
17     public class SynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress>
18     {
19         /// <summary>
20         /// <para>
21         /// Gets the constants value.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public LinksConstants<TLinkAddress> Constants
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get;
29         }
30
31         /// <summary>
32         /// <para>
33         /// Gets the sync root value.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         public ISynchronization SyncRoot
38         {
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             get;
41         }
42
43         /// <summary>
44         /// <para>
45         /// Gets the sync value.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public ILinks<TLinkAddress> Sync
50         {
51             [MethodImpl(MethodImplOptions.AggressiveInlining)]
52             get;
53         }
54
55         /// <summary>
56         /// <para>
57         /// Gets the unsync value.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         public ILinks<TLinkAddress> Unsync
62         {
63             [MethodImpl(MethodImplOptions.AggressiveInlining)]
64             get;
65         }
66
67         /// <summary>
68         /// <para>
69         /// Initializes a new <see cref="SynchronizedLinks"/> instance.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         /// <param name="links">
74         /// <para>A links.</para>
75         /// <para></para>
76         /// </param>
77         [MethodImpl(MethodImplOptions.AggressiveInlining)]
78         public SynchronizedLinks(ILinks<TLinkAddress> links) : this(new
            ↪ ReaderWriterLockSynchronization(), links) { }

```



```

79
80    /// <summary>
81    /// <para>
82    /// Initializes a new <see cref="SynchronizedLinks"/> instance.
83    /// </para>
84    /// <para></para>
85    /// </summary>
86    /// <param name="synchronization">
87    /// <para>A synchronization.</para>
88    /// <para></para>
89    /// </param>
90    /// <param name="links">
91    /// <para>A links.</para>
92    /// <para></para>
93    /// </param>
94    [MethodImpl(MethodImplOptions.AggressiveInlining)]
95    public SynchronizedLinks(ISynchronization synchronization, ILinks<TLinkAddress> links)
96    {
97        SyncRoot = synchronization;
98        Sync = this;
99        Unsync = links;
100        Constants = links.Constants;
101    }
102
103    /// <summary>
104    /// <para>
105    /// Counts the restriction.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    /// <param name="restriction">
110    /// <para>The restriction.</para>
111    /// <para></para>
112    /// </param>
113    /// <returns>
114    /// <para>The link address</para>
115    /// <para></para>
116    /// </returns>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    public TLinkAddress Count(ICollection<TLinkAddress>? restriction) =>
119        ↪ SyncRoot.ExecuteReadOperation(restriction, Unsync.Count);
120
121    /// <summary>
122    /// <para>
123    /// Eaches the handler.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="handler">
128    /// <para>The handler.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="restriction">
132    /// <para>The substitution.</para>
133    /// <para></para>
134    /// </param>
135    /// <returns>
136    /// <para>The link address</para>
137    /// <para></para>
138    /// </returns>
139    [MethodImpl(MethodImplOptions.AggressiveInlining)]
140    public TLinkAddress Each(ICollection<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
141        ↪ handler) => SyncRoot.ExecuteReadOperation(restriction, handler, Unsync.Each);
142
143    /// <summary>
144    /// <para>
145    /// Creates the substitution.
146    /// </para>
147    /// <para></para>
148    /// </summary>
149    /// <param name="substitution">
150    /// <para>The substitution.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The link address</para>
155    /// <para></para>
156    /// </returns>

```

```

155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
156     public TLinkAddress Create(ICollection<TLinkAddress>? substitution,
    ↪ WriteHandler<TLinkAddress>? handler) => SyncRoot.ExecuteWriteOperation(substitution,
    ↪ handler, Unsync.Create);

157
158     /// <summary>
159     /// <para>
160     /// Updates the substitution.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="restriction">
165     /// <para>The substitution.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="substitution">
169     /// <para>The substitution.</para>
170     /// <para></para>
171     /// </param>
172     /// <returns>
173     /// <para>The link address</para>
174     /// <para></para>
175     /// </returns>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     public TLinkAddress Update(ICollection<TLinkAddress>? restriction, ICollection<TLinkAddress>?
    ↪ substitution, WriteHandler<TLinkAddress>? handler) =>
    ↪ SyncRoot.ExecuteWriteOperation(restriction, substitution, handler, Unsync.Update);

178
179     /// <summary>
180     /// <para>
181     /// Deletes the substitution.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="restriction">
186     /// <para>The substitution.</para>
187     /// <para></para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     public TLinkAddress Delete(ICollection<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
    ↪ handler) => SyncRoot.ExecuteWriteOperation(restriction, handler, Unsync.Delete);

191
192     //public T Trigger(ICollection<T> restriction, Func<ICollection<T>, ICollection<T>, T> matchedHandler,
    ↪ ICollection<T> substitution, Func<ICollection<T>, ICollection<T>, T> substitutedHandler)
193     //{
194     //    if (restriction != null && substitution != null &&
    ↪ !substitution.EqualTo(restriction))
195     //        return SyncRoot.ExecuteWriteOperation(restriction, matchedHandler,
    ↪ substitution, substitutedHandler, Unsync.Trigger);
196
197     //    return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
    ↪ substitutedHandler, Unsync.Trigger);
198     //}
199 }
200 }

```

1.117 ./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Singletons;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 64 links extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class UInt64LinksExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// The instance.

```

```

22     /// </para>
23     /// <para></para>
24     /// </summary>
25     public static readonly LinksConstants<ulong> Constants =
        ↳ Default<LinksConstants<ulong>>.Instance;
26
27     /// <summary>
28     /// <para>
29     /// Determines whether any link is any.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     /// <param name="links">
34     /// <para>The links.</para>
35     /// <para></para>
36     /// </param>
37     /// <param name="sequence">
38     /// <para>The sequence.</para>
39     /// <para></para>
40     /// </param>
41     /// <returns>
42     /// <para>The bool</para>
43     /// <para></para>
44     /// </returns>
45     [MethodImpl(MethodImplOptions.AggressiveInlining)]
46     public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
47     {
48         if (sequence == null)
49         {
50             return false;
51         }
52         var constants = links.Constants;
53         for (var i = 0; i < sequence.Length; i++)
54         {
55             if (sequence[i] == constants.Any)
56             {
57                 return true;
58             }
59         }
60         return false;
61     }
62
63     /// <summary>
64     /// <para>
65     /// Formats the structure using the specified links.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="links">
70     /// <para>The links.</para>
71     /// <para></para>
72     /// </param>
73     /// <param name="linkIndex">
74     /// <para>The link index.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="isElement">
78     /// <para>The is element.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="renderIndex">
82     /// <para>The render index.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="renderDebug">
86     /// <para>The render debug.</para>
87     /// <para></para>
88     /// </param>
89     /// <returns>
90     /// <para>The string</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
        ↳ Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
        ↳ false)
95     {
96         var sb = new StringBuilder();

```

```

97     var visited = new HashSet<ulong>();
98     links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
99         ↪ innerSb.Append(link.Index), renderIndex, renderDebug);
100     return sb.ToString();
101 }
102
103 /// <summary>
104 /// <para>
105 /// Formats the structure using the specified links.
106 /// </para>
107 /// </summary>
108 /// <param name="links">
109 /// <para>The links.</para>
110 /// </param>
111 /// <param name="linkIndex">
112 /// <para>The link index.</para>
113 /// </param>
114 /// <param name="isElement">
115 /// <para>The is element.</para>
116 /// </param>
117 /// <param name="appendElement">
118 /// <para>The append element.</para>
119 /// </param>
120 /// <param name="renderIndex">
121 /// <para>The render index.</para>
122 /// </param>
123 /// <param name="renderDebug">
124 /// <para>The render debug.</para>
125 /// </param>
126 /// <returns>
127 /// <para>The string</para>
128 /// </returns>
129 [MethodImpl(MethodImplOptions.AggressiveInlining)]
130 public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
131     ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
132     ↪ bool renderIndex = false, bool renderDebug = false)
133 {
134     var sb = new StringBuilder();
135     var visited = new HashSet<ulong>();
136     links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
137         ↪ renderDebug);
138     return sb.ToString();
139 }
140
141 /// <summary>
142 /// <para>
143 /// Appends the structure using the specified links.
144 /// </para>
145 /// </summary>
146 /// <param name="links">
147 /// <para>The links.</para>
148 /// </param>
149 /// <param name="sb">
150 /// <para>The sb.</para>
151 /// </param>
152 /// <param name="visited">
153 /// <para>The visited.</para>
154 /// </param>
155 /// <param name="linkIndex">
156 /// <para>The link index.</para>
157 /// </param>
158 /// <param name="isElement">
159 /// <para>The is element.</para>
160 /// </param>
161 /// </summary>
162

```

```

170 /// </param>
171 /// <param name="appendElement">
172 /// <para>The append element.</para>
173 /// <para></para>
174 /// </param>
175 /// <param name="renderIndex">
176 /// <para>The render index.</para>
177 /// <para></para>
178 /// </param>
179 /// <param name="renderDebug">
180 /// <para>The render debug.</para>
181 /// <para></para>
182 /// </param>
183 /// <exception cref="ArgumentNullException">
184 /// <para></para>
185 /// <para></para>
186 /// </exception>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 public static void AppendStructure(this ILinks<ulong> links, StringBuilder sb,
    ↳ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
    ↳ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
    ↳ renderDebug = false)
189 {
190     if (sb == null)
191     {
192         throw new ArgumentNullException(nameof(sb));
193     }
194     if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
    ↳ Constants.Itself)
195     {
196         return;
197     }
198     if (links.Exists(linkIndex))
199     {
200         if (visited.Add(linkIndex))
201         {
202             sb.Append('(');
203             var link = new Link<ulong>(links.GetLink(linkIndex));
204             if (renderIndex)
205             {
206                 sb.Append(link.Index);
207                 sb.Append(':');
208             }
209             if (link.Source == link.Index)
210             {
211                 sb.Append(link.Index);
212             }
213             else
214             {
215                 var source = new Link<ulong>(links.GetLink(link.Source));
216                 if (isElement(source))
217                 {
218                     appendElement(sb, source);
219                 }
220                 else
221                 {
222                     links.AppendStructure(sb, visited, source.Index, isElement,
    ↳ appendElement, renderIndex);
223                 }
224             }
225             sb.Append(' ');
226             if (link.Target == link.Index)
227             {
228                 sb.Append(link.Index);
229             }
230             else
231             {
232                 var target = new Link<ulong>(links.GetLink(link.Target));
233                 if (isElement(target))
234                 {
235                     appendElement(sb, target);
236                 }
237                 else
238                 {
239                     links.AppendStructure(sb, visited, target.Index, isElement,
    ↳ appendElement, renderIndex);
240                 }
241             }

```

```

242         sb.Append(' ');
243     }
244     else
245     {
246         if (renderDebug)
247         {
248             sb.Append('*');
249         }
250         sb.Append(linkIndex);
251     }
252 }
253 else
254 {
255     if (renderDebug)
256     {
257         sb.Append('~');
258     }
259     sb.Append(linkIndex);
260 }
261 }
262 }
263 }

```

1.118 ./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.IO;
5  using System.Runtime.CompilerServices;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Platform.Disposables;
9  using Platform.Timestamps;
10 using Platform.Unsafe;
11 using Platform.IO;
12 using Platform.Data.Doublets.Decorators;
13 using Platform.Delegates;
14 using Platform.Exceptions;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets
19 {
20     /// <summary>
21     /// <para>
22     /// Represents the int 64 links transactions layer.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <seealso cref="LinksDisposableDecoratorBase{ulong}" />
27     public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<ulong> //-V3073
28     {
29         /// <remarks>
30         /// Альтернативные варианты хранения трансформации (элемента транзакции):
31         ///
32         /// private enum TransitionType
33         /// {
34         ///     Creation,
35         ///     UpdateOf,
36         ///     UpdateTo,
37         ///     Deletion
38         /// }
39         ///
40         /// private struct Transition
41         /// {
42         ///     public ulong TransactionId;
43         ///     public UniqueTimestamp Timestamp;
44         ///     public TransactionItemType Type;
45         ///     public Link Source;
46         ///     public Link Linker;
47         ///     public Link Target;
48         /// }
49         ///
50         /// Или
51         ///
52         /// public struct TransitionHeader
53         /// {
54         ///     public ulong TransactionIdCombined;
55         ///     public ulong TimestampCombined;

```

```

56     ///
57     ///     public ulong TransactionId
58     ///     {
59     ///         get
60     ///         {
61     ///             return (ulong) mask & TransactionIdCombined;
62     ///         }
63     ///     }
64     ///
65     ///     public UniqueTimestamp Timestamp
66     ///     {
67     ///         get
68     ///         {
69     ///             return (UniqueTimestamp)mask & TransactionIdCombined;
70     ///         }
71     ///     }
72     ///
73     ///     public TransactionItemType Type
74     ///     {
75     ///         get
76     ///         {
77     ///             // Использовать по одному биту из TransactionId и Timestamp,
78     ///             // для значения в 2 бита, которое представляет тип операции
79     ///             throw new NotImplementedException();
80     ///         }
81     ///     }
82     /// }
83     ///
84     /// private struct Transition
85     /// {
86     ///     public TransitionHeader Header;
87     ///     public Link Source;
88     ///     public Link Linker;
89     ///     public Link Target;
90     /// }
91     ///
92     /// </remarks>
93     public struct Transition : IEquatable<Transition>
94     {
95         /// <summary>
96         /// <para>
97         /// The size.
98         /// </para>
99         /// <para></para>
100        /// </summary>
101        public static readonly long Size = Structure<Transition>.Size;
102
103        /// <summary>
104        /// <para>
105        /// The transaction id.
106        /// </para>
107        /// <para></para>
108        /// </summary>
109        public readonly ulong TransactionId;
110        /// <summary>
111        /// <para>
112        /// The before.
113        /// </para>
114        /// <para></para>
115        /// </summary>
116        public readonly Link<ulong> Before;
117        /// <summary>
118        /// <para>
119        /// The after.
120        /// </para>
121        /// <para></para>
122        /// </summary>
123        public readonly Link<ulong> After;
124        /// <summary>
125        /// <para>
126        /// The timestamp.
127        /// </para>
128        /// <para></para>
129        /// </summary>
130        public readonly Timestamp Timestamp;
131
132        /// <summary>
133        /// <para>

```

```

134     /// Initializes a new <see cref="Transition"/> instance.
135     /// </para>
136     /// <para></para>
137     /// </summary>
138     /// <param name="uniqueTimestampFactory">
139     /// <para>A unique timestamp factory.</para>
140     /// <para></para>
141     /// </param>
142     /// <param name="transactionId">
143     /// <para>A transaction id.</para>
144     /// <para></para>
145     /// </param>
146     /// <param name="before">
147     /// <para>A before.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="after">
151     /// <para>A after.</para>
152     /// <para></para>
153     /// </param>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↪ transactionId, Link<ulong> before, Link<ulong> after)
156     {
157         TransactionId = transactionId;
158         Before = before;
159         After = after;
160         Timestamp = uniqueTimestampFactory.Create();
161     }
162
163     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↪ transactionId, IList<ulong> before, IList<ulong> after) :
        ↪ this(uniqueTimestampFactory, transactionId, new Link<ulong>(before), new
        ↪ Link<ulong>(after)) { }
164
165     /// <summary>
166     /// <para>
167     /// Initializes a new <see cref="Transition"/> instance.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="uniqueTimestampFactory">
172     /// <para>A unique timestamp factory.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="transactionId">
176     /// <para>A transaction id.</para>
177     /// <para></para>
178     /// </param>
179     /// <param name="before">
180     /// <para>A before.</para>
181     /// <para></para>
182     /// </param>
183     [MethodImpl(MethodImplOptions.AggressiveInlining)]
184     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↪ transactionId, Link<ulong> before) : this(uniqueTimestampFactory, transactionId,
        ↪ before, default) { }
185
186     /// <summary>
187     /// <para>
188     /// Initializes a new <see cref="Transition"/> instance.
189     /// </para>
190     /// <para></para>
191     /// </summary>
192     /// <param name="uniqueTimestampFactory">
193     /// <para>A unique timestamp factory.</para>
194     /// <para></para>
195     /// </param>
196     /// <param name="transactionId">
197     /// <para>A transaction id.</para>
198     /// <para></para>
199     /// </param>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↪ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
        ↪ }
202
203     /// <summary>

```



```

204     /// <para>
205     /// Returns the string.
206     /// </para>
207     /// <para></para>
208     /// </summary>
209     /// <returns>
210     /// <para>The string</para>
211     /// <para></para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     public override string ToString() => $"{Timestamp} {TransactionId}: {Before} =>
        ↳ {After}";
215
216     /// <summary>
217     /// <para>
218     /// Determines whether this instance equals.
219     /// </para>
220     /// <para></para>
221     /// </summary>
222     /// <param name="obj">
223     /// <para>The obj.</para>
224     /// <para></para>
225     /// </param>
226     /// <returns>
227     /// <para>The bool</para>
228     /// <para></para>
229     /// </returns>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     public override bool Equals(object obj) => obj is Transition transition ?
        ↳ Equals(transition) : false;
232
233     /// <summary>
234     /// <para>
235     /// Gets the hash code.
236     /// </para>
237     /// <para></para>
238     /// </summary>
239     /// <returns>
240     /// <para>The int</para>
241     /// <para></para>
242     /// </returns>
243     [MethodImpl(MethodImplOptions.AggressiveInlining)]
244     public override int GetHashCode() => (TransactionId, Before, After,
        ↳ Timestamp).GetHashCode();
245
246     /// <summary>
247     /// <para>
248     /// Determines whether this instance equals.
249     /// </para>
250     /// <para></para>
251     /// </summary>
252     /// <param name="other">
253     /// <para>The other.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
        ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;
262
263     [MethodImpl(MethodImplOptions.AggressiveInlining)]
264     public static bool operator ==(Transition left, Transition right) =>
        ↳ left.Equals(right);
265
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     public static bool operator !=(Transition left, Transition right) => !(left ==
        ↳ right);
268 }
269
270     /// <remarks>
271     /// Другие варианты реализации транзакций (атомарности):
272     /// 1. Разделение хранения значения связи ((Source Target) или (Source Linker
        ↳ Target)) и индексов.
273     /// 2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
        ↳ потребуется решить вопрос

```

```

274     /// со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
275     /// пересечениями идентификаторов.
276     /// Где хранить промежуточный список транзакций?
277     ///
278     /// В оперативной памяти:
279     /// Минусы:
280     /// 1. Может усложнить систему, если она будет функционировать самостоятельно,
281     /// так как нужно отдельно выделять память под список трансформаций.
282     /// 2. Выделенной оперативной памяти может не хватить, в том случае,
283     /// если транзакция использует слишком много трансформаций.
284     /// -> Можно использовать жёсткий диск для слишком длинных транзакций.
285     /// -> Максимальный размер списка трансформаций можно ограничить / задать
286     /// константой.
287     /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
288     /// создавая задержку.
289     ///
290     /// На жёстком диске:
291     /// Минусы:
292     /// 1. Длительный отклик, на запись каждой трансформации.
293     /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
294     /// -> Это может решаться упаковкой/исключением дублирующих операций.
295     /// -> Также это может решаться тем, что короткие транзакции вообще
296     /// не будут записываться в случае отката.
297     /// 3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
298     /// операции (трансформации)
299     /// будут записаны в лог.
300     ///
301     /// </remarks>
302     public class Transaction : DisposableBase
303     {
304         private readonly Queue<Transition> _transitions;
305         private readonly UInt64LinksTransactionsLayer _layer;
306         /// <summary>
307         /// <para>
308         /// Gets or sets the is committed value.
309         /// </para>
310         /// </summary>
311         public bool IsCommitted { get; private set; }
312         /// <summary>
313         /// <para>
314         /// Gets or sets the is reverted value.
315         /// </para>
316         /// </summary>
317         public bool IsReverted { get; private set; }
318         /// <summary>
319         /// <para>
320         /// Initializes a new <see cref="Transaction"/> instance.
321         /// </para>
322         /// </summary>
323         public Transaction(UInt64LinksTransactionsLayer layer)
324         {
325             _layer = layer;
326             if (_layer._currentTransactionId != 0)
327             {
328                 throw new NotSupportedException("Nested transactions not supported.");
329             }
330             IsCommitted = false;
331             IsReverted = false;
332             _transitions = new Queue<Transition>();
333             SetCurrentTransaction(layer, this);
334         }
335         /// <summary>
336         /// <para>

```

```

348     /// Commits this instance.
349     /// </para>
350     /// <para></para>
351     /// </summary>
352     [MethodImpl(MethodImplOptions.AggressiveInlining)]
353     public void Commit()
354     {
355         EnsureTransactionAllowsWriteOperations(this);
356         while (_transitions.Count > 0)
357         {
358             var transition = _transitions.Dequeue();
359             _layer._transitions.Enqueue(transition);
360         }
361         _layer._lastCommittedTransactionId = _layer._currentTransactionId;
362         IsCommitted = true;
363     }
364     [MethodImpl(MethodImplOptions.AggressiveInlining)]
365     private void Revert()
366     {
367         EnsureTransactionAllowsWriteOperations(this);
368         var transitionsToRevert = new Transition[_transitions.Count];
369         _transitions.CopyTo(transitionsToRevert, 0);
370         for (var i = transitionsToRevert.Length - 1; i >= 0; i--)
371         {
372             _layer.RevertTransition(transitionsToRevert[i]);
373         }
374         IsReverted = true;
375     }
376
377     /// <summary>
378     /// <para>
379     /// Sets the current transaction using the specified layer.
380     /// </para>
381     /// <para></para>
382     /// </summary>
383     /// <param name="layer">
384     /// <para>The layer.</para>
385     /// <para></para>
386     /// </param>
387     /// <param name="transaction">
388     /// <para>The transaction.</para>
389     /// <para></para>
390     /// </param>
391     [MethodImpl(MethodImplOptions.AggressiveInlining)]
392     public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
393     ↪ Transaction transaction)
394     {
395         layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
396         layer._currentTransactionTransitions = transaction._transitions;
397         layer._currentTransaction = transaction;
398     }
399
400     /// <summary>
401     /// <para>
402     /// Ensures the transaction allows write operations using the specified transaction.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="transaction">
407     /// <para>The transaction.</para>
408     /// <para></para>
409     /// </param>
410     /// <exception cref="InvalidOperationException">
411     /// <para>Transation is committed.</para>
412     /// <para></para>
413     /// </exception>
414     /// <exception cref="InvalidOperationException">
415     /// <para>Transation is reverted.</para>
416     /// <para></para>
417     /// </exception>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
420     {
421         if (transaction.IsReverted)
422         {
423             throw new InvalidOperationException("Transation is reverted.");
424         }
425         if (transaction.IsCommitted)

```

```

425         {
426             throw new InvalidOperationException("Transation is committed.");
427         }
428     }
429
430     /// <summary>
431     /// <para>
432     /// Disposes the manual.
433     /// </para>
434     /// <para></para>
435     /// </summary>
436     /// <param name="manual">
437     /// <para>The manual.</para>
438     /// <para></para>
439     /// </param>
440     /// <param name="wasDisposed">
441     /// <para>The was disposed.</para>
442     /// <para></para>
443     /// </param>
444     [MethodImpl(MethodImplOptions.AggressiveInlining)]
445     protected override void Dispose(bool manual, bool wasDisposed)
446     {
447         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
448         {
449             if (!IsCommitted && !IsReverted)
450             {
451                 Revert();
452             }
453             _layer.ResetCurrentTransation();
454         }
455     }
456 }
457
458 /// <summary>
459 /// <para>
460 /// The from seconds.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
465 private readonly string _logAddress;
466 private readonly FileStream _log;
467 private readonly Queue<Transition> _transitions;
468 private readonly UniqueTimestampFactory _uniqueTimestampFactory;
469 private Task _transitionsPusher;
470 private Transition _lastCommittedTransition;
471 private ulong _currentTransactionId;
472 private Queue<Transition> _currentTransactionTransitions;
473 private Transaction _currentTransaction;
474 private ulong _lastCommittedTransactionId;
475
476 /// <summary>
477 /// <para>
478 /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
479 /// </para>
480 /// <para></para>
481 /// </summary>
482 /// <param name="links">
483 /// <para>A links.</para>
484 /// <para></para>
485 /// </param>
486 /// <param name="logAddress">
487 /// <para>A log address.</para>
488 /// <para></para>
489 /// </param>
490 /// <exception cref="ArgumentNullException">
491 /// <para></para>
492 /// <para></para>
493 /// </exception>
494 /// <exception cref="NotSupportedException">
495 /// <para>Database is damaged, autorecovery is not supported yet.</para>
496 /// <para></para>
497 /// </exception>
498 [MethodImpl(MethodImplOptions.AggressiveInlining)]
499 public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
500     : base(links)
501 {
502     if (string.IsNullOrEmpty(logAddress))
503     {

```

```

504         throw new ArgumentNullException(nameof(logAddress));
505     }
506     // В первой строке файла хранится последняя закоммиченную транзакцию.
507     // При запуске это используется для проверки удачного закрытия файла лога.
508     // In the first line of the file the last committed transaction is stored.
509     // On startup, this is used to check that the log file is successfully closed.
510     var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
511     var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
512     if (!lastCommittedTransition.Equals(lastWrittenTransition))
513     {
514         Dispose();
515         throw new NotSupportedException("Database is damaged, autorecovery is not
516             ↳ supported yet.");
517     }
518     if (lastCommittedTransition == default)
519     {
520         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
521     }
522     _lastCommittedTransition = lastCommittedTransition;
523     // TODO: Think about a better way to calculate or store this value
524     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
525     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
526         ↳ x.TransactionId) : 0;
527     _uniqueTimestampFactory = new UniqueTimestampFactory();
528     _logAddress = logAddress;
529     _log = FileHelpers.Append(logAddress);
530     _transitions = new Queue<Transition>();
531     _transitionsPusher = new Task(TransitionsPusher);
532     _transitionsPusher.Start();
533 }
534
535 /// <summary>
536 /// <para>
537 /// Gets the link value using the specified link.
538 /// </para>
539 /// </summary>
540 /// <param name="link">
541 /// <para>The link.</para>
542 /// </param>
543 /// <returns>
544 /// <para>A list of ulong</para>
545 /// </returns>
546 [MethodImpl(MethodImplOptions.AggressiveInlining)]
547 public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
548
549 /// <summary>
550 /// <para>
551 /// Creates the substitution.
552 /// </para>
553 /// </summary>
554 /// <param name="substitution">
555 /// <para>The substitution.</para>
556 /// </param>
557 /// <returns>
558 /// <para>The created link index.</para>
559 /// </returns>
560 [MethodImpl(MethodImplOptions.AggressiveInlining)]
561 public override ulong Create(IList<ulong> substitution, WriteHandler<ulong> handler)
562 {
563     return _links.Create(new Link<ulong>(), (before, after) =>
564     {
565         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
566             ↳ new Link<ulong>(before), new Link<ulong>(after)));
567         return handler(before, after);
568     });
569 }
570
571 /// <summary>
572 /// <para>
573 /// Updates the substitution.
574 /// </para>
575 /// </summary>
576 /// <param name="substitution">
577 /// <para>The substitution.</para>
578 /// </param>

```

```

579     /// </summary>
580     /// <param name="restriction">
581     /// <para>The substitution.</para>
582     /// <para></para>
583     /// </param>
584     /// <param name="substitution">
585     /// <para>The substitution.</para>
586     /// <para></para>
587     /// </param>
588     /// <returns>
589     /// <para>The link index.</para>
590     /// <para></para>
591     /// </returns>
592     [MethodImpl(MethodImplOptions.AggressiveInlining)]
593     public override ulong Update(IList<ulong> restriction, IList<ulong> substitution,
594     ↪ WriteHandler<ulong> handler)
595     {
596         return _links.Update(restriction, substitution, (before, after) =>
597         {
598             CommitTransition(new Transition(_uniqueTimestampFactory,
599             ↪ _currentTransactionId, new Link<ulong>(before), new Link<ulong>(after)));
600             return handler(before, after);
601         });
602     }
603     /// <summary>
604     /// <para>
605     /// Deletes the substitution.
606     /// </para>
607     /// <para></para>
608     /// </summary>
609     /// <param name="restriction">
610     /// <para>The substitution.</para>
611     /// <para></para>
612     /// </param>
613     [MethodImpl(MethodImplOptions.AggressiveInlining)]
614     public override ulong Delete(IList<ulong> restriction, WriteHandler<ulong> handler)
615     {
616         var link = restriction[_constants.IndexPart];
617         return _links.Delete(restriction, (before, after) =>
618         {
619             CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
620             ↪ before, after));
621             return handler(before, after);
622         });
623     }
624     [MethodImpl(MethodImplOptions.AggressiveInlining)]
625     private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
626     ↪ _transitions;
627     [MethodImpl(MethodImplOptions.AggressiveInlining)]
628     private void CommitTransition(Transition transition)
629     {
630         if (_currentTransaction != null)
631         {
632             Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
633         }
634         var transitions = GetCurrentTransitions();
635         transitions.Enqueue(transition);
636     }
637     [MethodImpl(MethodImplOptions.AggressiveInlining)]
638     private void RevertTransition(Transition transition)
639     {
640         if (transition.After.IsNull()) // Revert Deletion with Creation
641         {
642             _links.Create();
643         }
644         else if (transition.Before.IsNull()) // Revert Creation with Deletion
645         {
646             _links.Delete(transition.After.Index);
647         }
648         else // Revert Update
649         {
650             _links.Update(new[] { transition.After.Index, transition.Before.Source,
651             ↪ transition.Before.Target });
652         }
653     }

```

```

651 [MethodImpl(MethodImplOptions.AggressiveInlining)]
652 private void ResetCurrentTransation()
653 {
654     _currentTransactionId = 0;
655     _currentTransactionTransitions = null;
656     _currentTransaction = null;
657 }
658 [MethodImpl(MethodImplOptions.AggressiveInlining)]
659 private void PushTransitions()
660 {
661     if (_log == null || _transitions == null)
662     {
663         return;
664     }
665     for (var i = 0; i < _transitions.Count; i++)
666     {
667         var transition = _transitions.Dequeue();
668
669         _log.Write(transition);
670         _lastCommittedTransition = transition;
671     }
672 }
673 [MethodImpl(MethodImplOptions.AggressiveInlining)]
674 private void TransitionsPusher()
675 {
676     while (!Disposable.IsDisposed && _transitionsPusher != null)
677     {
678         Thread.Sleep(DefaultPushDelay);
679         PushTransitions();
680     }
681 }
682
683 /// <summary>
684 /// <para>
685 /// Begins the transaction.
686 /// </para>
687 /// <para></para>
688 /// </summary>
689 /// <returns>
690 /// <para>The transaction</para>
691 /// <para></para>
692 /// </returns>
693 [MethodImpl(MethodImplOptions.AggressiveInlining)]
694 public Transaction BeginTransaction() => new Transaction(this);
695 [MethodImpl(MethodImplOptions.AggressiveInlining)]
696 private void DisposeTransitions()
697 {
698     try
699     {
700         var pusher = _transitionsPusher;
701         if (pusher != null)
702         {
703             _transitionsPusher = null;
704             pusher.Wait();
705         }
706         if (_transitions != null)
707         {
708             PushTransitions();
709         }
710         _log.DisposeIfPossible();
711         FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
712     }
713     catch (Exception ex)
714     {
715         ex.Ignore();
716     }
717 }
718
719 #region DisposalBase
720
721 /// <summary>
722 /// <para>
723 /// Disposes the manual.
724 /// </para>
725 /// <para></para>
726 /// </summary>
727 /// <param name="manual">
728 /// <para>The manual.</para>
729 /// <para></para>

```

```

730     /// </param>
731     /// <param name="wasDisposed">
732     /// <para>The was disposed.</para>
733     /// <para></para>
734     /// </param>
735     [MethodImpl(MethodImplOptions.AggressiveInlining)]
736     protected override void Dispose(bool manual, bool wasDisposed)
737     {
738         if (!wasDisposed)
739         {
740             DisposeTransitions();
741         }
742         base.Dispose(manual, wasDisposed);
743     }
744     #endregion
745 }
746 }
747 }

```

1.119 ./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs

```

1  using System;
2  using System.IO;
3  using Platform.Data.Doublets.Decorators;
4  using Xunit;
5
6  using Platform.Memory;
7
8  using Platform.Data.Doublets.Memory.United.Generic;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public static class GenericLinksTests
13     {
14         [Fact]
15         public static void CRUDTest()
16         {
17             Using<byte>(links => links.TestCRUDOperations());
18             Using<ushort>(links => links.TestCRUDOperations());
19             Using<uint>(links => links.TestCRUDOperations());
20             Using<ulong>(links => links.TestCRUDOperations());
21         }
22
23         [Fact]
24         public static void RawNumbersCRUDTest()
25         {
26             Using<byte>(links => links.TestRawNumbersCRUDOperations());
27             Using<ushort>(links => links.TestRawNumbersCRUDOperations());
28             Using<uint>(links => links.TestRawNumbersCRUDOperations());
29             Using<ulong>(links => links.TestRawNumbersCRUDOperations());
30         }
31
32         [Fact]
33         public static void MultipleRandomCreationsAndDeletionsTest()
34         {
35             Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
36                 ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
37                 ↪ implementation of tree cuts out 5 bits from the address space.
38             Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
39                 ↪ stMultipleRandomCreationsAndDeletions(100));
40             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
41                 ↪ MultipleRandomCreationsAndDeletions(100));
42             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
43                 ↪ tMultipleRandomCreationsAndDeletions(100));
44         }
45         private static void Using<TLink>(Action<ILinks<TLink>> action)
46         {
47             var unitedMemoryLinks = new UnitedMemoryLinks<TLink>(new
48                 ↪ HeapResizableDirectMemory());
49             using (var logFile = File.Open("linksLogger.txt", FileMode.Create, FileAccess.Write))
50             {
51                 LoggingDecorator<TLink> links = new(unitedMemoryLinks, logFile);
52                 action(links);
53             }
54
55             File.Delete("db.links");
56             using var ffiLinks = new FFI.UnitedMemoryLinks<TLink>("db.links");
57             action(ffiLinks);
58         }
59     }
60 }

```



```

53     }
54 }

```

1.120 ./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs

```

1  using System.IO;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Memory;
4  using Xunit;
5
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ILinksBasicTests
10     {
11         [Fact]
12         public static void DeleteAllUsages()
13         {
14             var mem = new HeapResizableDirectMemory();
15             var links = new UnitedMemoryLinks<uint>(mem);
16
17             var root = links.CreatePoint();
18
19             var a = links.CreatePoint();
20             var b = links.CreatePoint();
21
22             links.CreateAndUpdate(a, root);
23             links.CreateAndUpdate(b, root);
24
25             Assert.Equal(5U, links.Count());
26
27             links.DeleteAllUsages(root);
28
29             Assert.Equal(3U, links.Count());
30         }
31
32         [Fact]
33         public static void FfiDeleteAllUsages()
34         {
35             File.Delete("db.links");
36             var links = new FFI.UnitedMemoryLinks<uint>("db.links");
37
38             var root = links.CreatePoint();
39
40             var a = links.CreatePoint();
41             var b = links.CreatePoint();
42
43             links.CreateAndUpdate(a, root);
44             links.CreateAndUpdate(b, root);
45
46             Assert.Equal(5U, links.Count());
47
48             links.DeleteAllUsages(root);
49
50             Assert.Equal(3U, links.Count());
51         }
52     }
53 }

```

1.121 ./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs

```

1  using Xunit;
2
3  namespace Platform.Data.Doublets.Tests
4  {
5      public static class LinksConstantsTests
6      {
7          [Fact]
8          public static void ExternalReferencesTest()
9          {
10             LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
11                 ↪ (long.MaxValue + 1UL, ulong.MaxValue));
12
13             //var minimum = new Hybrid<ulong>(0, isExternal: true);
14             var minimum = new Hybrid<ulong>(1, isExternal: true);
15             var maximum = new Hybrid<ulong>(long.MaxValue, isExternal: true);
16
17             Assert.True(constants.IsExternalReference(minimum));
18             Assert.True(constants.IsExternalReference(maximum));
19         }
20     }

```

1.122 ./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs

```

1  using System.IO;
2  using Xunit;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.United.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ResizableDirectMemoryLinksTests
10     {
11         private static readonly LinksConstants<ulong> _constants =
12             ↳ Default<LinksConstants<ulong>>.Instance;
13
14         [Fact]
15         public static void BasicFileMappedMemoryTest()
16         {
17             var tempFilename = Path.GetTempFileName();
18             using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
19             {
20                 memoryAdapter.TestBasicMemoryOperations();
21             }
22             File.Delete(tempFilename);
23         }
24
25         [Fact]
26         public static void BasicHeapMemoryTest()
27         {
28             using (var memory = new
29                 ↳ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
30             using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
31                 ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
32             {
33                 memoryAdapter.TestBasicMemoryOperations();
34             }
35         }
36
37         private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
38         {
39             var link = memoryAdapter.Create();
40             memoryAdapter.Delete(link);
41         }
42
43         [Fact]
44         public static void NonexistentReferencesHeapMemoryTest()
45         {
46             using (var memory = new
47                 ↳ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
48             using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
49                 ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
50             {
51                 memoryAdapter.TestNonexistentReferences();
52             }
53         }
54
55         private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
56         {
57             var link = memoryAdapter.Create();
58             memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
59             var resultLink = _constants.Null;
60             memoryAdapter.Each(foundLink =>
61             {
62                 resultLink = foundLink[_constants.IndexPart];
63                 return _constants.Break;
64             }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
65             Assert.True(resultLink == link);
66             Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
67             memoryAdapter.Delete(link);
68         }
69     }
70 }

```

1.123 ./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs

```

1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Memory.United.Generic;
7  using Platform.Data.Doublets.Memory.United.Specific;
8

```

```

9 namespace Platform.Data.Doublets.Tests
10 {
11     public static class ScopeTests
12     {
13         [Fact]
14         public static void SingleDependencyTest()
15         {
16             using (var scope = new Scope())
17             {
18                 scope.IncludeAssemblyOf<IMemory>();
19                 var instance = scope.Use<IDirectMemory>();
20                 Assert.IsType<HeapResizableDirectMemory>(instance);
21             }
22         }
23
24         [Fact]
25         public static void CascadeDependencyTest()
26         {
27             using (var scope = new Scope())
28             {
29                 scope.Include<TemporaryFileMappedResizableDirectMemory>();
30                 scope.Include<UInt64UnitedMemoryLinks>();
31                 var instance = scope.Use<ILinks<ulong>>();
32                 Assert.IsType<UInt64UnitedMemoryLinks>(instance);
33             }
34         }
35
36         [Fact(Skip = "Would be fixed later.")]
37         public static void FullAutoResolutionTest()
38         {
39             using (var scope = new Scope(autoInclude: true, autoExplore: true))
40             {
41                 var instance = scope.Use<UInt64Links>();
42                 Assert.IsType<UInt64Links>(instance);
43             }
44         }
45
46         [Fact]
47         public static void TypeParametersTest()
48         {
49             using (var scope = new Scope<Types<HeapResizableDirectMemory,
50 ↪ UnitedMemoryLinks<ulong>>>())
51             {
52                 var links = scope.Use<ILinks<ulong>>();
53                 Assert.IsType<UnitedMemoryLinks<ulong>>(links);
54             }
55         }
56     }

```

1.124 ./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs

```

1 using System;
2 using Xunit;
3 using Platform.Memory;
4 using Platform.Data.Doublets.Memory.Split.Generic;
5 using Platform.Data.Doublets.Memory;
6
7 namespace Platform.Data.Doublets.Tests
8 {
9     public unsafe static class SplitMemoryGenericLinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using<byte>(links => links.TestCRUDOperations());
15             Using<ushort>(links => links.TestCRUDOperations());
16             Using<uint>(links => links.TestCRUDOperations());
17             Using<ulong>(links => links.TestCRUDOperations());
18         }
19
20         [Fact]
21         public static void RawNumbersCRUDTest()
22         {
23             UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
24             UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
25             UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
26             UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
27         }
28     }

```

```

29 [Fact]
30 public static void MultipleRandomCreationsAndDeletionsTest()
31 {
32     Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
        ↳ implementation of tree cuts out 5 bits from the address space.
33     Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
        ↳ stMultipleRandomCreationsAndDeletions(100));
34     Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(100));
35     Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
        ↳ tMultipleRandomCreationsAndDeletions(100));
36 }
37 private static void Using<TLink>(Action<ILinks<TLink>> action)
38 {
39     using (var dataMemory = new HeapResizableDirectMemory())
40     using (var indexMemory = new HeapResizableDirectMemory())
41     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
42     {
43         action(memory);
44     }
45 }
46 private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
47 {
48     var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
49     using (var dataMemory = new HeapResizableDirectMemory())
50     using (var indexMemory = new HeapResizableDirectMemory())
51     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory,
        ↳ SplitMemoryLinks<TLink>.DefaultLinksSizeStep, constants))
52     {
53         action(memory);
54     }
55 }
56 }
57 }

```

1.125 ./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt32;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryUInt32LinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using(links => links.TestCRUDOperations());
15         }
16
17         [Fact]
18         public static void RawNumbersCRUDTest()
19         {
20             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21         }
22
23         [Fact]
24         public static void MultipleRandomCreationsAndDeletionsTest()
25         {
26             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
        ↳ leRandomCreationsAndDeletions(500));
27         }
28         private static void Using(Action<ILinks<TLink>> action)
29         {
30             using (var dataMemory = new HeapResizableDirectMemory())
31             using (var indexMemory = new HeapResizableDirectMemory())
32             using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory))
33             {
34                 action(memory);
35             }
36         }
37         private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
38         {
39             var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
40             using (var dataMemory = new HeapResizableDirectMemory())

```

```

41         using (var indexMemory = new HeapResizableDirectMemory())
42         using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
43             ↪ UInt32SplitMemoryLinks.DefaultLinksSizeStep, contants))
44         {
45             action(memory);
46         }
47     }
48 }

```

1.126 ./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt64;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryUInt64LinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using(links => links.TestCRUDOperations());
15         }
16
17         [Fact]
18         public static void RawNumbersCRUDTest()
19         {
20             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21         }
22
23         [Fact]
24         public static void MultipleRandomCreationsAndDeletionsTest()
25         {
26             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip_
27                 ↪ leRandomCreationsAndDeletions(500));
28         }
29         private static void Using(Action<ILinks<TLink>> action)
30         {
31             using (var dataMemory = new HeapResizableDirectMemory())
32             using (var indexMemory = new HeapResizableDirectMemory())
33             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
34             {
35                 action(memory);
36             }
37         }
38         private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
39         {
40             var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
41             using (var dataMemory = new HeapResizableDirectMemory())
42             using (var indexMemory = new HeapResizableDirectMemory())
43             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
44                 ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, contants))
45             {
46                 action(memory);
47             }
48         }
49     }
50 }

```

1.127 ./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class TestExtensions
12     {
13         public static void TestCRUDOperations<T>(this ILinks<T> links)
14         {
15             var constants = links.Constants;
16         }
17     }
18 }

```

```

17     var equalityComparer = EqualityComparer<T>.Default;
18
19     var zero = default(T);
20     var one = Arithmetic.Increment(zero);
21
22     // Create Link
23     Assert.True(equalityComparer.Equals(links.Count(), zero));
24
25     var setter = new Setter<T>(constants.Null);
26     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
27
28     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
29
30     var linkAddress = links.Create();
31
32     var link = new Link<T>(links.GetLink(linkAddress));
33
34     Assert.True(link.Count == 3);
35     Assert.True(equalityComparer.Equals(link.Index, linkAddress));
36     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
37     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
38
39     Assert.True(equalityComparer.Equals(links.Count(), one));
40
41     // Get first link
42     setter = new Setter<T>(constants.Null);
43     links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
44
45     Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
46
47     // Update link to reference itself
48     links.Update(linkAddress, linkAddress, linkAddress);
49
50     link = new Link<T>(links.GetLink(linkAddress));
51
52     Assert.True(equalityComparer.Equals(link.Source, linkAddress));
53     Assert.True(equalityComparer.Equals(link.Target, linkAddress));
54
55     // Update link to reference null (prepare for delete)
56     var updated = links.Update(linkAddress, constants.Null, constants.Null);
57
58     Assert.True(equalityComparer.Equals(updated, linkAddress));
59
60     link = new Link<T>(links.GetLink(linkAddress));
61
62     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
63     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
64
65     // Delete link
66     links.Delete(linkAddress);
67
68     Assert.True(equalityComparer.Equals(links.Count(), zero));
69
70     setter = new Setter<T>(constants.Null);
71     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
72
73     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
74 }
75
76 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
77 {
78     // Constants
79     var constants = links.Constants;
80     var equalityComparer = EqualityComparer<T>.Default;
81
82     var zero = default(T);
83     var one = Arithmetic.Increment(zero);
84     var two = Arithmetic.Increment(one);
85
86     var h106E = new Hybrid<T>(106L, isExternal: true);
87     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
88     var h108E = new Hybrid<T>(-108L);
89
90     Assert.Equal(106L, h106E.AbsoluteValue);
91     Assert.Equal(107L, h107E.AbsoluteValue);
92     Assert.Equal(108L, h108E.AbsoluteValue);
93
94     // Create Link (External -> External)
95     var linkAddress1 = links.Create();
96

```

```

97     links.Update(linkAddress1, h106E, h108E);
98
99     var link1 = new Link<T>(links.GetLink(linkAddress1));
100
101     Assert.True(equalityComparer.Equals(link1.Source, h106E));
102     Assert.True(equalityComparer.Equals(link1.Target, h108E));
103
104     // Create Link (Internal -> External)
105     var linkAddress2 = links.Create();
106
107     links.Update(linkAddress2, linkAddress1, h108E);
108
109     var link2 = new Link<T>(links.GetLink(linkAddress2));
110
111     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
112     Assert.True(equalityComparer.Equals(link2.Target, h108E));
113
114     // Create Link (Internal -> Internal)
115     var linkAddress3 = links.Create();
116
117     links.Update(linkAddress3, linkAddress1, linkAddress2);
118
119     var link3 = new Link<T>(links.GetLink(linkAddress3));
120
121     Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
122     Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
123
124     // Search for created link
125     var setter1 = new Setter<T>(constants.Null);
126     links.Each(h106E, h108E, setter1.SetAndReturnFalse);
127
128     Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
129
130     // Search for nonexistent link
131     var setter2 = new Setter<T>(constants.Null);
132     links.Each(h106E, h107E, setter2.SetAndReturnFalse);
133
134     Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
135
136     // Update link to reference null (prepare for delete)
137     var updated = links.Update(linkAddress3, constants.Null, constants.Null);
138
139     Assert.True(equalityComparer.Equals(updated, linkAddress3));
140
141     link3 = new Link<T>(links.GetLink(linkAddress3));
142
143     Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
144     Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
145
146     // Delete link
147     links.Delete(linkAddress3);
148
149     Assert.True(equalityComparer.Equals(links.Count(), two));
150
151     var setter3 = new Setter<T>(constants.Null);
152     links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
153
154     Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
155 }
156
157 public static void TestMultipleCreationsAndDeletions<TLink>(this ILinks<TLink> links,
158     ↪ int numberOfOperations)
159 {
160     for (int i = 0; i < numberOfOperations; i++)
161     {
162         links.Create();
163     }
164     for (int i = 0; i < numberOfOperations; i++)
165     {
166         links.Delete(links.Count());
167     }
168 }
169
170 public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
171     ↪ links, int maximumOperationsPerCycle)
172 {
173     var comparer = Comparer<TLink>.Default;
174     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
175     var uint64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
176     for (var N = 1; N < maximumOperationsPerCycle; N++)

```

```

175     {
176         var random = new System.Random(N);
177         var created = 0UL;
178         var deleted = 0UL;
179         for (var i = 0; i < N; i++)
180         {
181             var linksCount = addressToUInt64Converter.Convert(links.Count());
182             var createPoint = random.NextBoolean();
183             if (linksCount >= 2 && createPoint)
184             {
185                 var linksAddressRange = new Range<ulong>(1, linksCount);
186                 TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
187                     ↪ ddressRange));
188                 TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
189                     ↪ ddressRange));
190                 ↪ //-V3086
191                 var resultLink = links.GetOrCreate(source, target);
192                 if (comparer.Compare(resultLink,
193                     ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
194                 {
195                     created++;
196                 }
197             }
198             else
199             {
200                 links.Create();
201                 created++;
202             }
203         }
204         Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
205         for (var i = 0; i < N; i++)
206         {
207             TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
208             if (links.Exists(link))
209             {
210                 links.Delete(link);
211                 deleted++;
212             }
213         }
214         Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
215     }
216 }

```

1.128 ./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs

```

1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Numbers.Raw;
4  using Platform.Memory;
5  using Platform.Numbers;
6  using Xunit;
7  using Xunit.Abstractions;
8  using TLink = System.UInt64;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public class UInt64LinksExtensionsTests
13     {
14         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
15             ↪ Platform.IO.TemporaryFile());
16
17         public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
18         {
19             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
20                 ↪ true);
21             return new UnitedMemoryLinks<TLink>(new
22                 ↪ FileMappedResizableDirectMemory(dataDBFilename),
23                 ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
24                 ↪ IndexTreeType.Default);
25         }
26         [Fact]
27         public void FormatStructureWithExternalReferenceTest()
28         {
29             ILinks<TLink> links = CreateLinks();
30             TLink zero = default;
31             var one = Arithmetic.Increment(zero);
32             var markerIndex = one;
33             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);

```



```

29         var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
        ↪ markerIndex));
30         AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
31         var numberAddress = addressToNumberConverter.Convert(1);
32         var numberLink = links.GetOrCreate(numberMarker, numberAddress);
33         var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
        ↪ true);
34         Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
35     }
36 }
37 }

```

1.129 ./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLink = System.UInt32;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt32LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24
25         [Fact]
26         public static void MultipleRandomCreationsAndDeletionsTest()
27         {
28             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
        ↪ leRandomCreationsAndDeletions(100));
29         }
30         private static void Using(Action<ILinks<TLink>> action)
31         {
32             using (var scope = new Scope<Types<HeapResizableDirectMemory,
        ↪ UInt32UnitedMemoryLinks>>())
33             {
34                 action(scope.Use<ILinks<TLink>>());
35             }
36         }
37     }
38 }

```

1.130 ./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLink = System.UInt64;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt64LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24     }

```

```

25     [Fact]
26     public static void MultipleRandomCreationsAndDeletionsTest()
27     {
28         Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
29     }
30     private static void Using(Action<ILinks<TLink>> action)
31     {
32         using (var scope = new Scope<Types<HeapResizableDirectMemory,
33             ↳ UInt64UnitedMemoryLinks>>())
34         {
35             action(scope.Use<ILinks<TLink>>());
36         }
37     }
38 }

```

Index

`./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs`, 456
`./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs`, 457
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 457
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 458
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 458
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 459
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 460
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 461
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 461
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 464
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 465
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs`, 465
`./csharp/Platform.Data.Doublets/CriterionMatchers/TargetMatcher.cs`, 1
`./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs`, 1
`./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs`, 2
`./csharp/Platform.Data.Doublets/Decorators/LinksDecoratorBase.cs`, 3
`./csharp/Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs`, 5
`./csharp/Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs`, 7
`./csharp/Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.cs`, 8
`./csharp/Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs`, 9
`./csharp/Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.cs`, 10
`./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessResolver.cs`, 11
`./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessValidator.cs`, 12
`./csharp/Platform.Data.Doublets/Decorators/LinksUsagesValidator.cs`, 13
`./csharp/Platform.Data.Doublets/Decorators/LoggingDecorator.cs`, 14
`./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs`, 15
`./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs`, 16
`./csharp/Platform.Data.Doublets/Decorators/UInt64Links.cs`, 17
`./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs`, 19
`./csharp/Platform.Data.Doublets/Doublet.cs`, 25
`./csharp/Platform.Data.Doublets/DoubletComparer.cs`, 27
`./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs`, 28
`./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs`, 30
`./csharp/Platform.Data.Doublets/ILinks.cs`, 39
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 39
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 59
`./csharp/Platform.Data.Doublets/Link.cs`, 59
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 67
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 67
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 68
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 69
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 70
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 71
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 73
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 79
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 86
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 90
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 94
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 98
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 102
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs`, 107
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs`, 113
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 118
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 121
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 125
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 129
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 132
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 136
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 154
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 157
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 158
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 160
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 166
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 172
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 176

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 180
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 184
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 187
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 193
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 199
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 200
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 203
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 207
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 211
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 214
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 221
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 222
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 228
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 234
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 238
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 241
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 245
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 249
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 255
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 260
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 262
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 265
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 269
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 272
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 276
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 283
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 284
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 293
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 299
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 306
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 311
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 315
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 318
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 323
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 327
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 331
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 333
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 347
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 350
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 351
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 357
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 362
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 366
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 370
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 374
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 377
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 383
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 384
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 391
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 397
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 402
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 408
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 411
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 415
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 420
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 424
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 428
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 434
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 435
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 436
./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 437
./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 439

./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 439
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 442
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 446